dqMan 5

User manual

for Documentum Content Server

fme

Table of contents

0. SYSTEM REQUIREMENTS 5

1. PREFACE 6

1.1. License Agreement 7
1.2. Supported Documentum Clients 9
1.3. License Level 10
1.4. New Features in Release 5 11
1.4.1. Former Xtended Features 11
1.4.2. All Editions 12
1.4.3. Xtended Edition 14
1.5. Installing dqMan on a removable media 15

2. SESSIONS 16

2.1. Session management 16
2.2. Login 17
2.3. Docbroker Availability 18

3. SESSION WINDOW 19

3.1. The DQL View 20
3.1.1. Working with DQL Queries 21
3.1.2. Syntax assistance 21
3.1.3. Query by Label 22
3.1.4. The Result list 23
3.1.5. The Result List Inline Editor 26
3.1.6. The Attribute Filter 28
3.1.7. The Message Area 30
3.1.8. The Describe Command 31
3.1.9. DQL Favorites 33
3.1.10. Favorite menu 36
3.1.11. Template Queries 37
3.1.12. Formating DQL Queries 40
3.1.13. Repeated Queries 42
3.2. The API View 43
3.3. The DQL and API Script View 45
3.3.1. Executing Scripts 46
3.3.2. The Script Navigator 47
3.3.3. Error Handling 47
3.3.4. Using Variables in API Scripts 48
3.3.5. Using Script Commands in API Scripts 49
3.3.6. Logging 51

4. CREATING A DUMP OF AN OBJECT 53

5. SEARCH 54

6. THE HISTORY 55

7. THE REPOSITORY NAVIGATOR 56

7.1. The Cabinet/Folder Tree 57
7.2. The Object Type/Registered Table Tree 58
7.3. The Group Tree 59
7.4. The Favorites Tree 60

8. THE OBJECT NAVIGATOR 61

9. PROCESSING ATTRIBUTE DEFINITIONS 68

9.1. Dump Attribute 68
9.2. Edit Attribute 69
9.3. Copy Attribute 70

10. VIEWING TEXTFILES 73

10.1. File View 73
10.2. Tail View 73

11. THE SCRIPT GENERATOR 74

11.1. Script Templates 75
11.1.1. Placeholders 75
11.1.2. Processing instructions 76
11.1.3. Examples 77
11.2. Managing Script Templates 78
11.3. The History 78
11.4. Script Types and Options 78
11.5. Adding Data Sources 79
11.6. Generating a Script 81
11.7. Executing a Script 81
11.8. Type dependent Script Templates 81

12. IMPORT 83

12.1. Import of CSV or Text Files 84
12.2. Import of Excel Files 85
12.3. Import from Clipboard 85

13. BATCH OBJECT CREATION 86

14. COMPARE DATA 87

14.1. The „Compare Data“ Function 87
14.1.1. Settings 87
14.1.2. Data Columns 88
14.1.3. Results 89
14.1.4. Working with Marked Records 90
14.2. The Object Compare List 92

15. SCRIPT ASSISTANTS 94

15.1. Dump & Load 94
15.1.1. Dump entire docbase 94
15.1.2. Dump Specific Objects 95
15.1.3. Load Docbase 95
15.2. Copy Registered Table 96
15.3. Change Objecttype 101

16. OPTIONS 104

16.1. General 104
16.1.1. Settings 104
16.1.2. Login 104
16.1.3. Shared Favorites 105
16.1.4. Documentation 105
16.2. API Command View 106
16.3. DQL Command View 107
16.3.1. Settings 107
16.3.2. Colors 108
16.3.3. Syntax Highlighting 108
16.3.4. Grid Hints 109
16.3.5. Repository Navigator 112

16.3.5.1. Cabinets/Folder Tree ... 112
16.3.5.2. Object Type Tree ... 113
16.3.5.3. Group Tree ... 113
16.4. Script Views ... 114
16.5. Functions ... 115
16.5.1. Object Functions ... 115
16.5.2. Session Functions ... 128
16.6. Miscellaneous ... 129
16.6.1. Docbroker Availability ... 129
16.6.2. Docbasic Interpreter ... 130
16.6.3. Tracing ... 130

17. APPENDIX A – UNSUPPORTED COMMANDS AND DQMAN SYNTAX EXTENSIONS ... 131

17.1. The API Command: filename ... 131
17.2. The DQL Administration Method: set_session_dd_locale ... 131
17.3. The DQL Administration Method: count_sessions ... 132
17.4. Script Commands and Variables ... 132

18. APPENDIX B - PLUGINS ... 133

18.1. The DMCL.INI / dfc Properties Editor ... 134
18.2. The ACL Viewer ... 136
18.3. dqMan Analyzer ... 140
18.4. Content Restore ... 142

0. System Requirements

Operation System: All current Windows plattforms

Please make sure that you are granted **local administration permissions** to install dqMan **on Windows 7** or **Windows Server 2008**.

Documentum 4i and 5	Documentum Client dmcl40.dll
Documentum 6	Documentum Client dmcl40.dll or Documentum DFC 6.x and Documentum dmcl.dll

1. Preface

The languages DQL and API provided by EMC Documentum offer a wide spectrum of possibilities to administrate, maintain and program Documentum's Content Server. With dqMan we deliver an integrated interface that allows fast and effective use of these languages.

This manual contains information and instructions for using dqMan 5. Readers are expected to be familiar with Documentum technologies, especially DQL and API.

dqMan is delivered without any further support, as described in the license agreement. Nevertheless we are looking forward to your feedback. Please send any feedback, questions, feature requests or bug reports to dqman@fme.de.

1.1. License Agreement

License Agreement

Please read the following terms carefully before downloading and installing the Software. By downloading or installing this copy, you agree to the terms of this license agreement. Other Terms and Conditions are not accepted. The use of the Software is exclusively subject to the following terms. If you do not agree to any of the terms of this agreement, you may not download and install this Software.

I. Subject of This Agreement

1. fme AG grants the licensee the non-exclusive right to use the Software for an indefinite time at no cost.
2. The Software is provided for download on the fme AG website in the Internet.

II. Licensee's Right of Use

1. This copy of the Software is exclusively for use at one workplace.
2. The Software is copyrighted. The copyright remains with fme AG. The licensee is not permitted to assign or distribute sublicenses.
3. The licensee is entitled to produce copies as long as they exclusively serve for the protection of data. The licensee is not permitted to produce copies of this Software for any other purposes, to reproduce the Software in any other way and/or to make the Software available to third parties.
4. The licensee is not permitted to make any changes to the Software, especially to develop it any further, to retranslate it or to extract portions from it. The licensee may not change or remove the copyright or any other protection rights within the Software.
5. In case the license terms change, the licensee shall be informed thereof by fme AG and shall accept these changes. Failure to accept these changes will cause the licensee to lose this license.

III. Warranty

The licensee is aware of the fact that, given the current state of technology, errors in software programs cannot be excluded and that it is impossible to create data-processing programs so that they function flawlessly under any possible application conditions and under any requirements on the part of the licensee with any programs of third parties. Guaranties regarding certain properties or the usability for individual applications intended by the licensee are not made.

In view of the free provision of the Software, liability for any defects in quality and title is limited to malicious non-disclosure on the part of fme.

IV. Liability

Liability on the part of fme is excluded due to its free provision of the Software, except for liability pursuant to product liability law; that is, for intent or malice aforethought, for gross negligence, for harm to life, limb or health, for malicious non-disclosure of a defect or for violating basic contractual obligations. Liability claims for violation of material contractual obligations are limited to the typical foreseeable damages, unless intent or gross negligence is involved. This does not imply in any way a reversal of onus of proof against the licensee.

V. Termination

1. fme reserves the right to withdraw the license from the licensee via extraordinary termination for violation of this license agreement.
2. The licensee is entitled to terminate this license agreement at any time.
3. In case of termination or loss of the license, the licensee is obligated to return the Software, including its documentation and all copies.

VI. Miscellaneous Terms

1. fme AG reserves the right to raise license fees for new versions of this Software.
2. If license fees are raised, the licensee shall have the right to extraordinary termination without notice.
3. Place of performance is Braunschweig, Germany.
4. This agreement is subject to Federal German Law under exclusion of the United Nations Convention of Contracts for the International Sale of Goods (CISG).
5. Place of venue is Braunschweig, Germany, inasmuch as a valid agreement can be reached to that effect.

VII. Severability Clause

In the event that one of the above terms is or becomes invalid, the parties shall agree to an effective settlement that best approximates the original economic intent.

1.2. Supported Documentum Clients

dqMan supports two Documentum clients:

dmcl40.dll is a native dmcl client. It can be used for Documentum 4 and 5 content servers. As Documentum 6 is fully compatible to clients of the edition 5, it can be used for Documentum 6 as well.

Please make sure, that the dmcl40.dll from any Documentum 5 Edition is installed and reachable by system path.

Connecting to Documentum using dmcl40.dll will bypass installed TBOs.

dmcl.dll is a dmcl emulator installed with Documentum 6, translating API commands to dfc. Therefore, a valid Documentum DFC 6.x must be installed. The dmcl.dll must be reachable by system path.

Connecting to Documentum using dmcl.dll with dfc will trigger TBOs the same way as normal dfc commands do.

On start-up dqMan searches at first for a dmcl.dll. If it is found and can be initialized dqMan will use it as Documentum client. Otherwise dqMan will search for a dmcl40.dll. If no valid client can be found, an error message is displayed.

The start-up order can be modified by passing the parameter –ForceDMCL to the dqMan.exe. In this case dqMan looks for a dmcl40.dll first. This parameter will be passed to other dqMan Suite applications like djMan or daMan if started from the dqMan toolbar.

Example:

dqMan -ForceDMCL

The About dialog (Help → About) contains information about the used client.

1.3. License Level

There are two license levels available for dqMan, the Standard Version and the Xtended Version. These levels differ in functionality. Functions that are only available in the Xtended Version are marked with a red bar at the left border.

This function is only available in the Xtended Version.

1.4. New Features in Release 5

1.4.1. Former Xtended Features

The following features are now also available in the dqMan standard version:

DQL Query:

Unformat Query

Highlight History

History entries are highlighted during browsing

API Command

Browse for filenames in API Command area (context menu)

Session Alias

Select an alias name for a session

1.4.2. All Editions

DQL Query

Syntax check for unclosed brackets (), [] or single quotes ".

Display of server response time in status bar.

Describe command can be stopped during execution

Support for undocumented adminstative function: execute count_sessions

Template queries and object functions:

Use cached queries option for dynamic value assistances.

Message Area

Logging options: Auto Clear Log, Log Query Results in Fixed Width or CSV format.

API Scripts

Script command #$Sleep supported in API scripts.

Command DEFINE no longer valid

Increment and decrement variables containing numbers,

e.g. %LoopIndex%++ or %PositionID%--

Repository Navigator

New icons for roles

Script Generator

Autocomplete can be activated with <Ctrl>+<Space>.

All attributes of an object can be referenced. The attribute does not need to be part of the underlying result list anymore.

New option for DQL scripts: Auto-escape single quotes

Object functions

Syntax highlighting added.

Value assistance can be invoked manually using <Ctrl>+<Space>.

Indenting with function code is now allowed.

Increment and decrement variables containing numbers,

e.g. %LoopIndex%++ or %PositionID%--

Grid hints

Value assistance can be invoked manually using <Ctrl>+<Space>.

New dqMan DQL administration method

set_session_dd_locale added to set locales in DQL script.

Dump and Load

Optionally stop DMClean and DMFilescan jobs during load operation

Destroy dump-/loadobject otional

Change Object Type

Auto-escape of DQL reserved words

DQL History, Functions and Favorites

Repository names are replaced by repository IDs for assigning DQL history, functions and favorites to certain repositories

Compare Data

Autofind function for corresponding attribute

Import from File

Ignore Empty Lines option for import from CSV and clipboard

1.4.3. Xtended Edition

Repository Navigator

- Registered Tables can be listed in types tab.

Copy Registered Table

- Available from Repository Navigator
- Name of target table can be changed
- Hide system tables function on table selection page
- Auto-escape of DQL reserved words
- Auto-split SQL Create Table statements if too long

Attribute Copy Script

- Check for references to source object type in value assistances
- Auto-escape of DQL reserved words

Attribute Editor

- Auto-escape of DQL reserved words

Object Navigator

- Quick-Picks with Dropdown and History
- Quick-Picks result list reduced to unique objects
- Copy values with doubleclick
- Add to Compare List Function added
- Destroy Folder Function (recursive) added
- Create ticketed user session Function added

dqMan Analyzer

- Template queries: support cached queries for value assistances
- Analyzes can be limited to repositories by supplying a repository ID

1.5. Installing dqMan on a removable media

dqMan can be used from a removable medium, like an USB stick.

To install dqMan on a removable medium, you can either install it directly using the dqMan installer or copy the entire dqMan program directory (default is c:\program files\dqMan). In any case, the installer will register a few file types, like API or DQL, for the dqMan application. dqMan itself makes no use of the windows registry.

To create a fully usable copy of dqMan, also copy the dmcl40.dll and the dmcl.ini files to the dqMan program directory on the removable medium. This will allow you to execute dqMan from any computer, without having a Documentum client installed on it.

2. Sessions

Sessions are always bound to a session window. Every session window has its own session. Closing a session window will terminate the session, but terminating a session will not close the session window.

dqMan allows you to open as many sessions as needed, restricted only by settings in the client or server configuration.

2.1. Session management

Sessions are managed using the Session menu or the toolbar. Available functions are:

New:	Opens a new session window with login dialog.
Copy:	Creates a copy of the current session window and logs on automatically using the same user credentials.
Trace:	Activates/deactivates DCML tracing.
Disconnect:	Terminates a session without closing the session window. Button description changes to Connect.
Quick Login:	If valid logins exist in the login history, you can open a dropdown list with available history entries by clicking on the down arrow beside the New or Connect button.
Refresh Value lists:	Reloads session dependent value lists, like users, groups, or object types.
Locales:	Switches current session locale to locale selected from submenu.
Session Alias:	Sets an alias name for a session. This name is used instead of default session name <user>@<repository> wherever it is possible.

2.2. Login

There are different ways to open a session:

- Opening a session window
- Selecting Connect from the session menu or the toolbar
- Executing an API or DQL command, that needs a session, without having a valid session. (Local commands like *getservermap*, *getdocbrokermap* or *connect* can be executed without a session)

Whenever a session is needed but not available the login dialog will appear.

Logins can be saved in a history for repeated use, either with or without password. Saved logins are available through the *History* button. Behaviour can be changed in the dqMan Options, see *16.1.2 Login*.

2.3. Docbroker Availability

At start-up dqMan can check whether a docbroker host is available. Please refer to the “Options” section for configuring this feature.

If the docbroker host is not available, an error window is displayed, which shows all docbrokers configured in the dmcl.ini/dfc.properties file along with their status.

If you select a new docbroker from the docbroker history or add a new docbroker manually, you can use the test button to check for availability. Your changes will be stored in the dmcl.ini/dfc.properties file, if you continue with OK.

3. Session Window

The Session Window has 4 functional views:

DQL:	Single DQL command view
API:	Single API command view
DQL Script:	DQL script view
API Script:	API script view

These 4 views can be used independently. Changing the view does not delete any commands or results. During script debugging, commands in other views can be executed without stopping the debug process.

Views can be selected from the View menu or from the toolbar.

Figure 1 – Session Window Views

3.1. The DQL View

Single DQL queries can be executed in the DQL View.

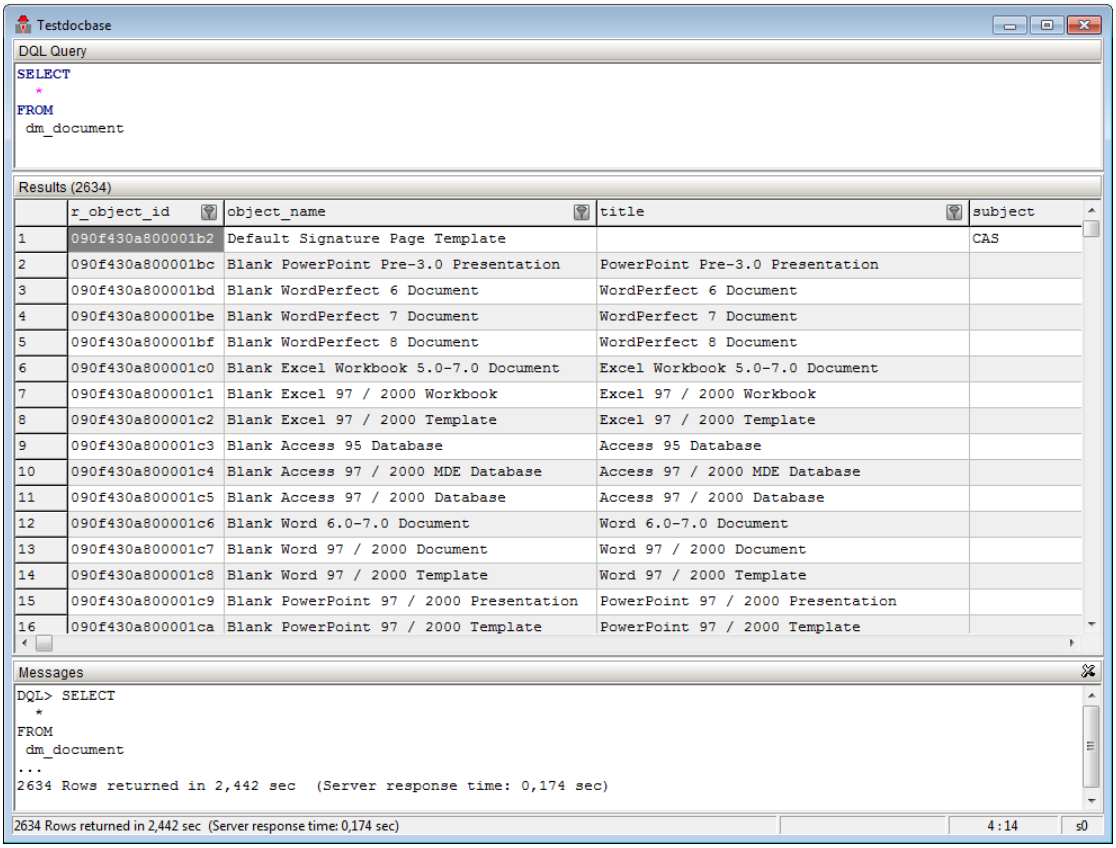

Testdocbase
DQL Query
SELECT
*
FROM
dm_document
Results (2634)
r_object_id
object_name
title
subject
1 090f430a800001b2 Default Signature Page Template CAS
2 090f430a800001bc Blank PowerPoint Pre-3.0 Presentation PowerPoint Pre-3.0 Presentation
3 090f430a800001bd Blank WordPerfect 6 Document WordPerfect 6 Document
4 090f430a800001be Blank WordPerfect 7 Document WordPerfect 7 Document
5 090f430a800001bf Blank WordPerfect 8 Document WordPerfect 8 Document
6 090f430a800001c0 Blank Excel Workbook 5.0-7.0 Document Excel Workbook 5.0-7.0 Document
7 090f430a800001c1 Blank Excel 97 / 2000 Workbook Excel 97 / 2000 Workbook
8 090f430a800001c2 Blank Excel 97 / 2000 Template Excel 97 / 2000 Template
9 090f430a800001c3 Blank Access 95 Database Access 95 Database
10 090f430a800001c4 Blank Access 97 / 2000 MDE Database Access 97 / 2000 MDE Database
11 090f430a800001c5 Blank Access 97 / 2000 Database Access 97 / 2000 Database
12 090f430a800001c6 Blank Word 6.0-7.0 Document Word 6.0-7.0 Document
13 090f430a800001c7 Blank Word 97 / 2000 Document Word 97 / 2000 Document
14 090f430a800001c8 Blank Word 97 / 2000 Template Word 97 / 2000 Template
15 090f430a800001c9 Blank PowerPoint 97 / 2000 Presentation PowerPoint 97 / 2000 Presentation
16 090f430a800001ca Blank PowerPoint 97 / 2000 Template PowerPoint 97 / 2000 Template
Messages
DQL> SELECT
*
FROM
dm_document
...
2634 Rows returned in 2,442 sec (Server response time: 0,174 sec)
2634 Rows returned in 2,442 sec (Server response time: 0,174 sec)
4 : 14

Figure 2 - The DQL View

To execute a DQL query press <F9> or <Shift>+<Enter>, select *Run* from the Edit menu or the context menu or click on the *Run* button in the toolbar. A running query can be stopped by clicking on the *Stop* button in the toolbar.

Double-clicking a cell in the result grid will copy the value into the clipboard. If the <Ctrl> key is pressed while double-clicking, the cell value will be immediately pasted into the command area at the current cursor position.

Messages in the status bar can be copied into the clipboard by double clicking as well. Moving the mouse cursor over the status bar will display the current message in a hint window, if it does not fit into the message bar area.

The Messages area contains information about the last executed action. It can also show the query result.

The status bar shows the complete execution time of the query, including transferring query results to dqMan. The pure server response time without data transfer is given as *Server response time*.

3.1.1. Working with DQL Queries

In the input area one or more DQL queries can be entered. If the query is executed, all content of the input area is regarded as one query and transferred to the Documentum server for the execution. However if a part of a query is selected, only this part is executed.

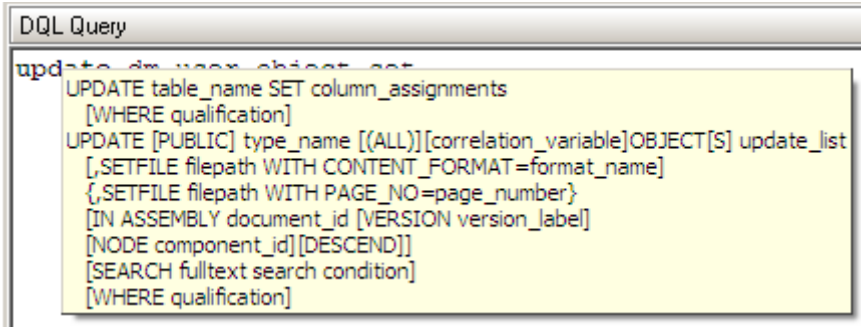
If more than one query is entered in the input area, they can be separated by a semicolon at the end of each query. Setting the text cursor on one query selects this query for execution.

Additionally comments are possible. Each line beginning with the # character or with // is regarded as comment line.

3.1.2. Syntax assistance

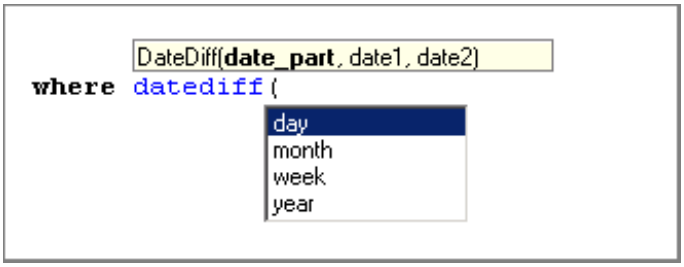
Creating DQL queries is supported by a number of syntax assistance functions:

Command syntax:



Click on a DQL command and wait for the hint to view the correct command syntax.

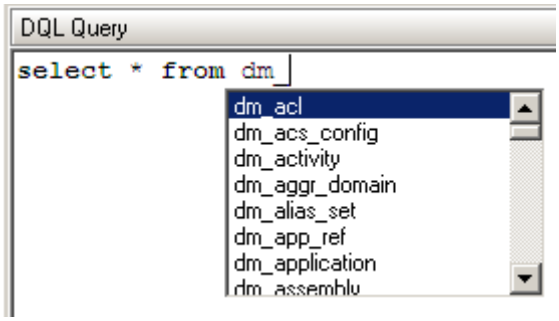
Function syntax:



The syntax of functions, e.g. Dateadd, Datediff etc. will appear automatically while typing the DQL command.

This syntax assistance can be activated also at any time pressing the key F1.

Auto completion:



Several dynamic values, like object names, group names etc., or allowed function values are faded in automatically in form of a selection list.

Values from selection lists can be copied by pressing <Enter>.

The selection list can be manually activated anywhere in the command area pressing <Ctrl>+<Space>. The list is filled context dependent with the values likely at the current position. If none of the pre-defined lists applies, the list is filled with the possible attribute names.

Some of the selection lists are read dynamically from the repository when opening the session. The values of these lists can be subject to changes by other activities that will not be recognized by dqMan. In this case the lists can be reloaded selecting *Refresh Valuelists* from the Session menu.

3.1.3. Query by Label

In DQL queries attribute labels of any locale can be used instead of the original attribute name. dqMan will replace labels with the attribute names at execution. Attribute labels must be placed in square brackets within the query.

```
Select r_object_id,object_name,[ACL Name],[Application type]
from dm_document where …
```

3.1.4. The Result list

In the result list the results of the DQL queries are represented in tabular form.

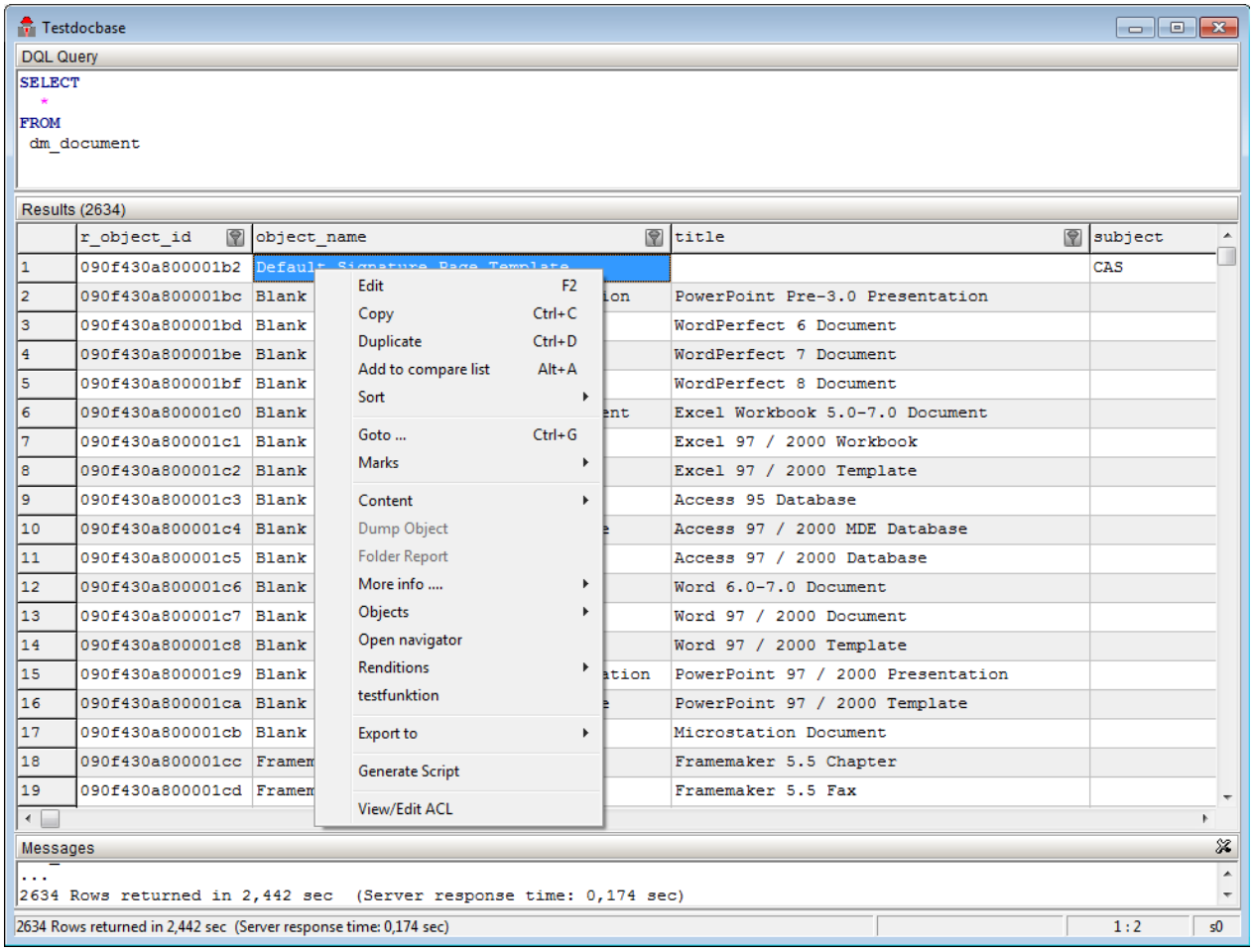
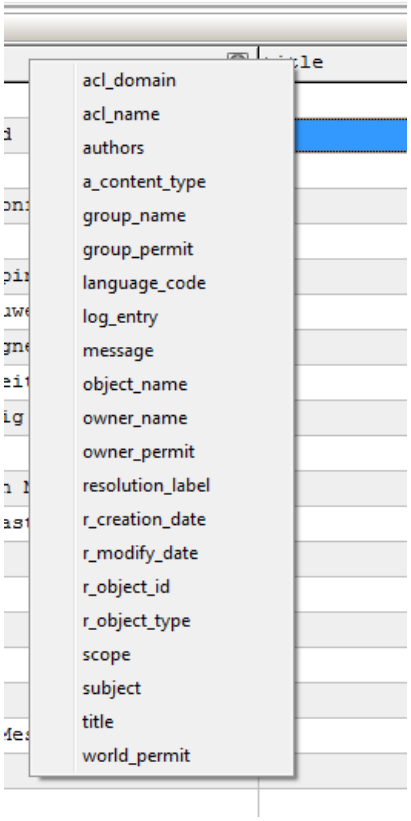


Figure 3 – Result list

The columns and rows of the result list can be sized in width and height. Columns can be moved using the mouse.

Within the result list search functions are available. A search over the entire table takes place by means of the function Search from the menu Edit. Typing one or several initial letters within one column will scroll to the first value matching.

Two context menus are available for the result list, which are activated with the right mouse button.



On the column heading line, an alphabetically sorted list of all available columns appears.

Clicking on an item scrolls this column into the visible range of the list.

On the data area, a list of functions is faded in for object processing. Available functions are:

Edit

Starts the Result List Inline Editor. *(→ 3.1.5 The Result List Inline Editor)*

Copy

This function copies selected values into the clipboard.

Rows will be separated with a linefeed and a carriage return character; columns will be separated with tabs.

Duplicate

Duplicate copies all selected values in a separate data grid.

Add to compare list

The selected objects are added to the Object Compare List.

Sort

Sorts the result data ascending or descending by the selected column.

Goto ...

The function Goto… enables the fast change to a line or a column for the result list.

If a number is entered, the list will be scrolled vertically, if a column header is entered or selected from the dropdown list, it will be scrolled horizontally.

Marks

Marks are used to select different objects in the result list. Marks can be toggled or scrolled to the next or previous marked object. Marks are mainly used by the *Compare Data* function, but can also be set manually.

Object functions

In the following section of the context menu user-defined functions for objects in the result list are offered (e.g. Change ACL, Dump Object, Last SQL statement, Request PDF rendition …). Object functions are administrated in the Options Dialog.

Object functions are offered depending on the selected type of document and the individual configuration.

Export to

Exports the selected cells of the result list to an Excel or CSV formatted file. To use the Excel export, Excel must be installed.

More than one Excel tab sheet will be used, if the amount of exported data exceeds the Excel limitations of 65535 lines and 256 columns.

Generate Script:

Opens the Script Generator, that lets you generate DQL or API scripts from the selected data.

Additional type dependent script templates will also be available, if the appropriate object type is selected.

Plugin Functions

If dqMan Plugins are installed, that supply object related functions, these functions will be displayed at the end of the context menu.

3.1.5. The Result List Inline Editor

Values can be edited directly in the result list, if the attribute r_object_id is contained in the results. To save any changes they need to be confirmed with the key <Enter>. Pressing <ESC> or moving the focus to any other control will close the editor without saving.

Single Attributes

For single attributes a standard input field will be created in the corresponding cell of the grid.

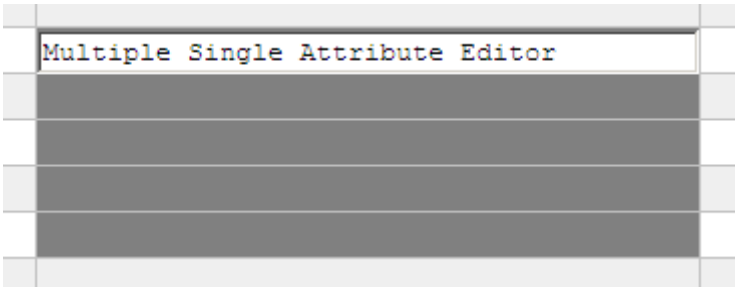
For some attributes a drop-down list is shown, which contains possible values. This list is filled either with the values of a data dictionary static value assistance, or with manually configured values from the file *dqManValuelists.xml*.

Repeating Attributes

Repeating attributes can be edited within a small sub grid. As DQL results do not show the correct order of repeating attribute values, the contents of this list can differ from the displayed values in the result list. The sub grid offers an own context menu for inserting, deleting or reordering values of the repeating attribute.

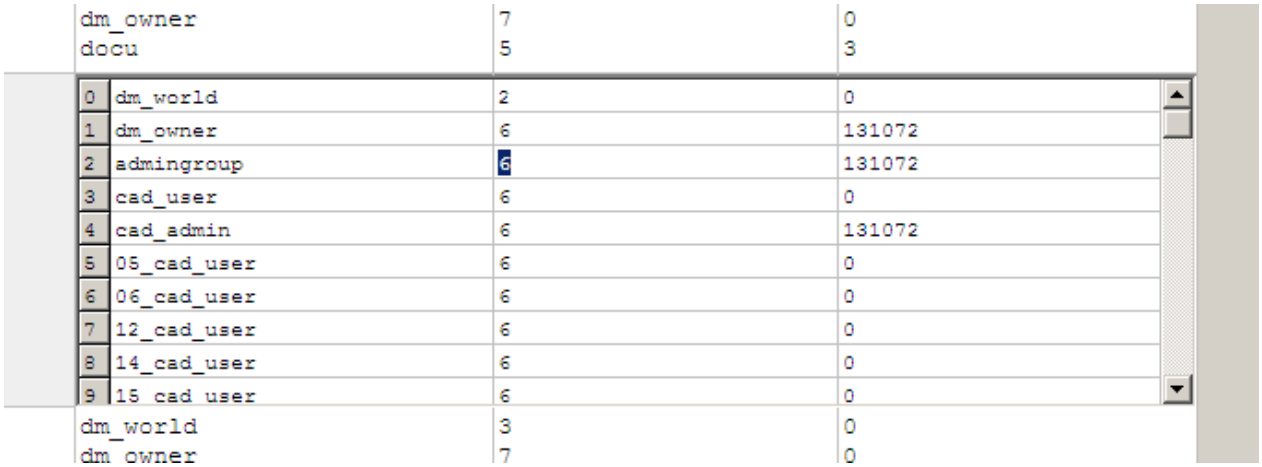
Multiple Single Attributes

If the result list contains more than one result, a single attribute of multiple objects can be edited simultaneously. Select more than one value in a column and start the editor.



Linked Repeating Attributes

Linked repeating attributes of one object such as **r_accessor_name, r_accessor_permit and r_accessor_xpermit** of **dm_acl** objects can be edited in a multicolumn sub grid. Select more than one column of repeating attributes and start the editor.



	dm_owner	7	0
	docu	5	3
0	dm_world	2	0
1	dm_owner	6	131072
2	admingroup	6	131072
3	cad_user	6	0
4	cad_admin	6	131072
5	05_cad_user	6	0
6	06_cad_user	6	0
7	12_cad_user	6	0
8	14_cad_user	6	0
9	15_cad_user	6	0
	dm_world	3	0
	dm_owner	7	0

3.1.6. The Attribute Filter

By using the attribute filter, query results can be filtered by the queried values. It can be opened for each column. Filters can be used for multiple columns. They will be combined by an “and” operation.

To open an attribute filter for a column, click on the button in the column header. The filter dialog opens, showing filter options depending on column data type.

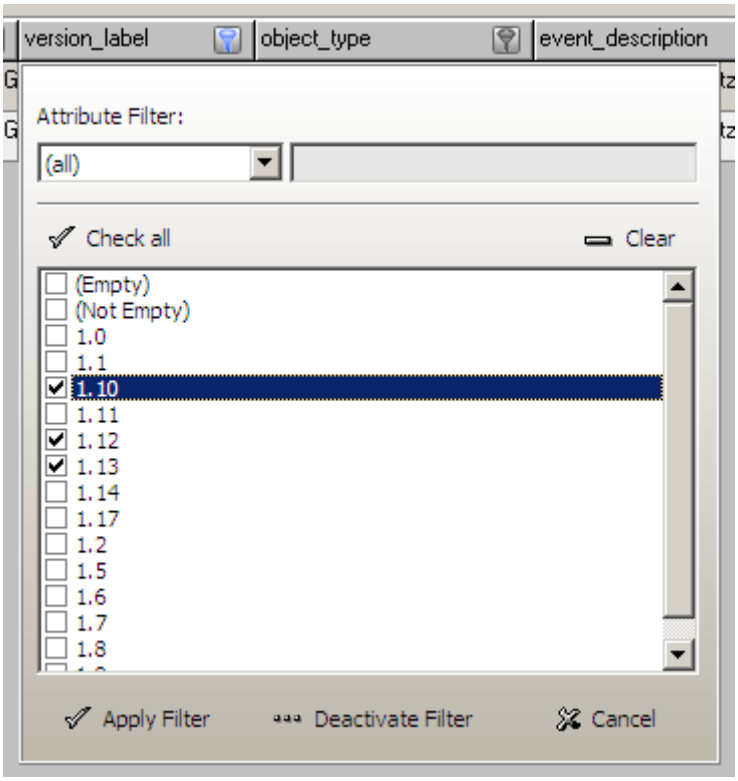


Figure 4 – Attribute Filter

Click on Apply Filter to activate the filter after selecting the filter options. If a filter is active, the filter button in the column header and the row numbers are highlighted.

Additionally, all filter buttons in column headers have a context menu.

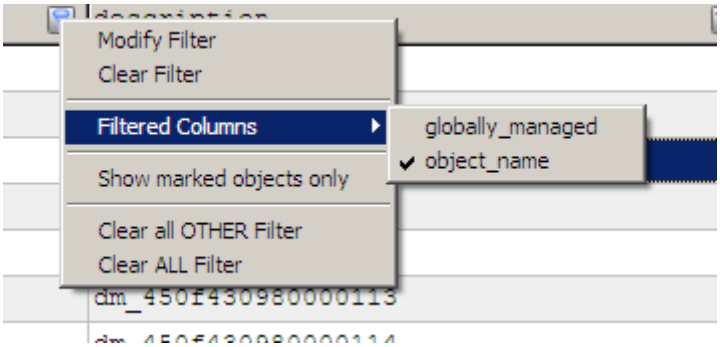


Figure 5 – Attribute Filter context menu

Modify Filter	Opens the filter dialog.
Clear Filter	Clears the selected filter and updates the list.
Filtered Columns	Shows a list of all active filters in the list. Selecting a column scrolls the list to the corresponding column.
Show marked objects only	Limits the filter to marked objects, e.g. after using Compare Data function.
Clear all OTHER Filter	Clears all other filters except for the selected one.
Clear ALL Filter	Clears all filter.

If at least one filter is active, the top left cell of the result grid also shows a filter button with an active context menu. This menu has one additional function: “Generate Predicate”. Executing this function generates a predicate (part of a where clause) from the current filter settings and copies it into the clipboard. It can be added to the current query to replace the filter settings.

A predicate (in the example shown above) would look like:

```
(Upper(object_name) like 'DM%') and
(globally_managed in (0)) and
(acl_class < 3)
```

3.1.7. The Message Area

The Message Area logs the executed DQL commands and their results. The context menu offers some settings to influence the behaviour:

Copy	Copies selected text to clipboard
Clear	Clears message Area
Word Wrap	Toggles word wrap setting

Settings

Auto Clear Log:	Message Area can be cleared before executing a query (setting is ignored for auto repeated query).
Log Query Results	DQL results will also be logged in the message area, not only messages.
CSV Format for Query Results:	Results of DQL queries will be logged in CSV format.
Fixed Width for Query Results:	Results of DQL queries will be logged as list with fixed width. Values can be truncated, if too long.

The Message Area can be activated from the View menu.

3.1.8. The Describe Command

The Describe Command is entered in the DQL input window and displays all attributes and selected characteristics for an object type.

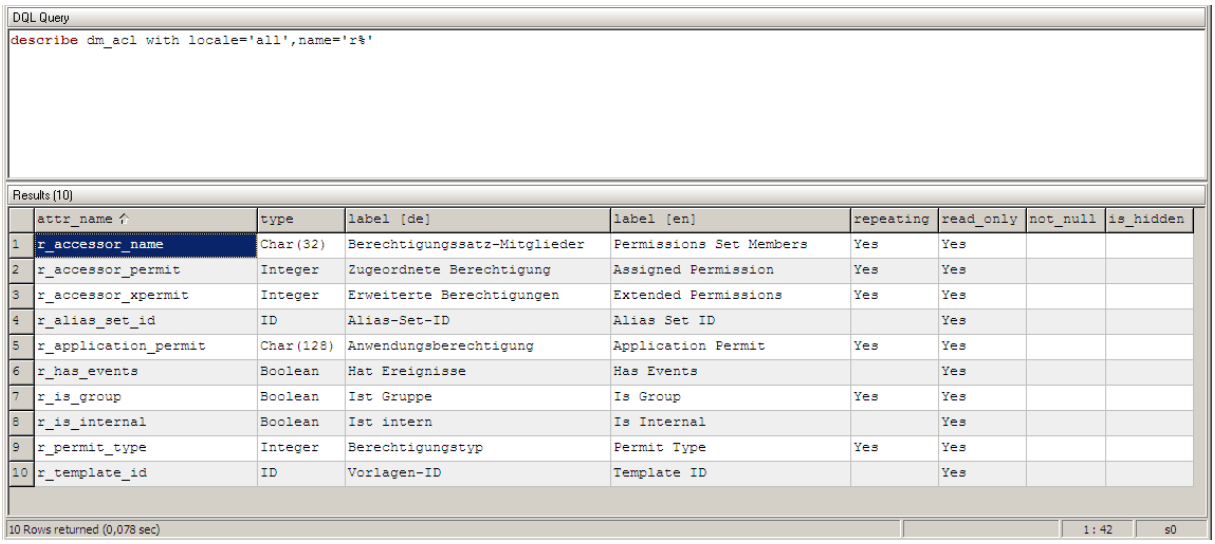
Syntax:

```
Describe <type> [all|custom] [with
name='<name>'][,locale='<locale>{,<locale>}|all']
Describe <table> [name='<identifier>']
```

Arguments:

`<type>`	Object type, like dm_document, dm_acl or custom types.
`[all|custom]`	`all` lists all attributes including all inherited ones, `custom` lists all customized attributes, including parent objects. If no argument is supplied, only the type related attributes will be listed. (Available for object types only)
`<table>`	Name of a registered table, e.g. dm_dbo.dm_queue or dm_queue.
`name='<name>'`	Part of or complete attribute name and/or column name. Wildcards are "*" or "%".
`locale='locale'`	Locale name for the attribute labels. Multiple locales can be separated by comma. locale='all' displays all attribute labels of all available locales. (Available for object types only)

The attribute labels can be edited directly in the result list, using the edit function from the context menu. Changed labels will be published immediately. Due to local caching the displayed labels can differ after editing, until the local cache is updated.



DQL Query

```
describe dm_acl with locale='all',name='r%'
```

Results (10)

	attr_name	type	label [de]	label [en]	repeating	read_only	not_null	is_hidden
1	r_accessor_name	Char(32)	Berechtigungssatz-Mitglieder	Permissions Set Members	Yes	Yes		
2	r_accessor_permit	Integer	Zugeordnete Berechtigung	Assigned Permission	Yes	Yes		
3	r_accessor_xpermit	Integer	Erweiterte Berechtigungen	Extended Permissions	Yes	Yes		
4	r_alias_set_id	ID	Alias-Set-ID	Alias Set ID		Yes		
5	r_application_permit	Char(128)	Anwendungsberechtigung	Application Permit	Yes	Yes		
6	r_has_events	Boolean	Hat Ereignisse	Has Events		Yes		
7	r_is_group	Boolean	Ist Gruppe	Is Group	Yes	Yes		
8	r_is_internal	Boolean	Ist intern	Is Internal		Yes		
9	r_permit_type	Integer	Berechtigungstyp	Permit Type	Yes	Yes		
10	r_template_id	ID	Vorlagen-ID	Template ID		Yes		

10 Rows returned (0,078 sec) 1 : 42 s0

Figure 6 - Describe Command

Examples:

```
Describe dm_acl
```

Lists all attributes of type dm_acl.

```
Describe dm_acl all
```

Lists all attributes of type dm_acl, including inherited attributes.

```
Describe dm_document custom with name='i_%'
```

Lists all customized attributes of type dm_acl that begin with the characters „i_”.

```
Describe dm_dbo.dm_queue with name='a%'
```

List column definitions of the registered table dm_queue that begin with the character “a“.

The Describe Command is also available from the context menu of the input area. The function is enabled if the context menu is opened on the name of an object type or a registered table. This call corresponds to the command:

```
Describe <type or table> all
```

3.1.9. DQL Favorites

DQL queries can be saved to the favorites view. Every query can be attributed to enable easy and fast access.

The favorites can be opened from the Edit menu. To store a query call *Edit -> Store in Favorites*. Either the whole content of the input area, or if anything is selected the selected content only, will be saved as favorite.

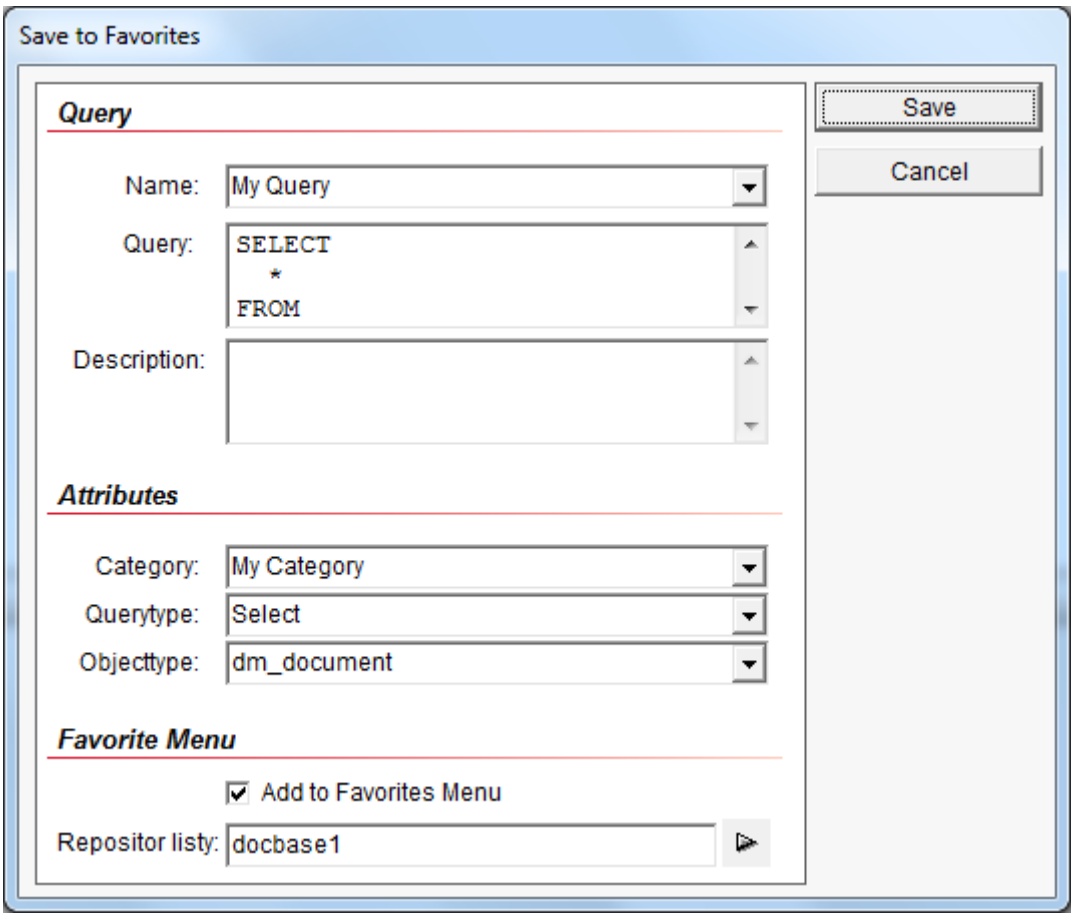


Figure 7 – Store in Favorites

Name:

Name of the query.

Query:

The DQL query.

Description:

Description of the query.

Category:

Freely selectable category, e.g. "Workflow", "ACL", "Users and groups". If a query should appear in the *Favorite menu*, the category is mandatory as queries will be grouped by their categories there.

Query type:

Type of the query, e.g. Select, Update, Execute etc.

Object type:

Object type contained in the query.

Favorite Menu:

To add a query to the Favorite Menu, check „*Add to Favorites Menu*".

Docbase List enables the allocation of queries to one or more repositories. An exclamation mark placed in front of the repository name, e.g. „!testdocbase", hides this query if this repository is active.

Repositories can either be added manually, or selected from a repository list by clicking on . Multiple repository names must be separated with a comma.

Repositories are stored by their repository id. If you are working with clustered content servers or use full qualified repository names, every repository needs to be listed only pnce.

To load a query from the favorites, call *Edit -> Load from Favorites.*

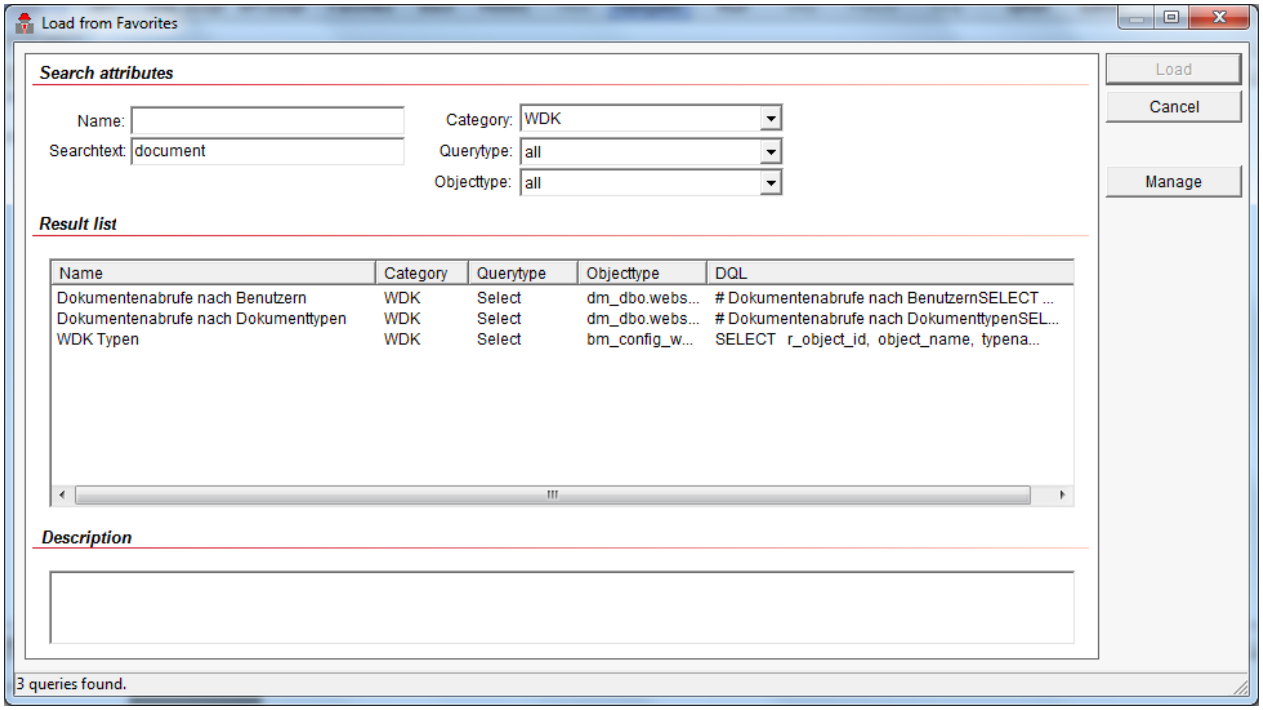
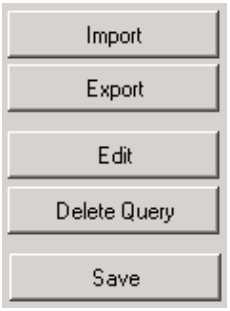


Figure 8 – Load from Favorites

On the top of the window is a search filter. The search criteria can be combined freely. *Searchtext* does a “full_text” search in all available favorite attributes. In the center part of the window all queries are displayed, that correspond to the entered filter criteria. A double click on a query copies it to the DQL View input area. If a description exists for a query, it will be displayed in the lower part of the window if the query is selected.

Additionally, the favorites can be managed within this dialog by clicking on the button *Manage*. A separate button list will appear.



Click on *Import* to import existing favorites from your file system.

Click on *Export* to export all selected queries to the file system.

Click on *Edit* to edit the selected query.

Click on *Delete Query* to remove all selected queries.

Click on *Save* to save your changes.

3.1.10. Favorite menu

Frequently used DQL queries can be made available in the favorite menu by activating the checkbox A*dd to Favorites* in the *Store to Favorites* function.

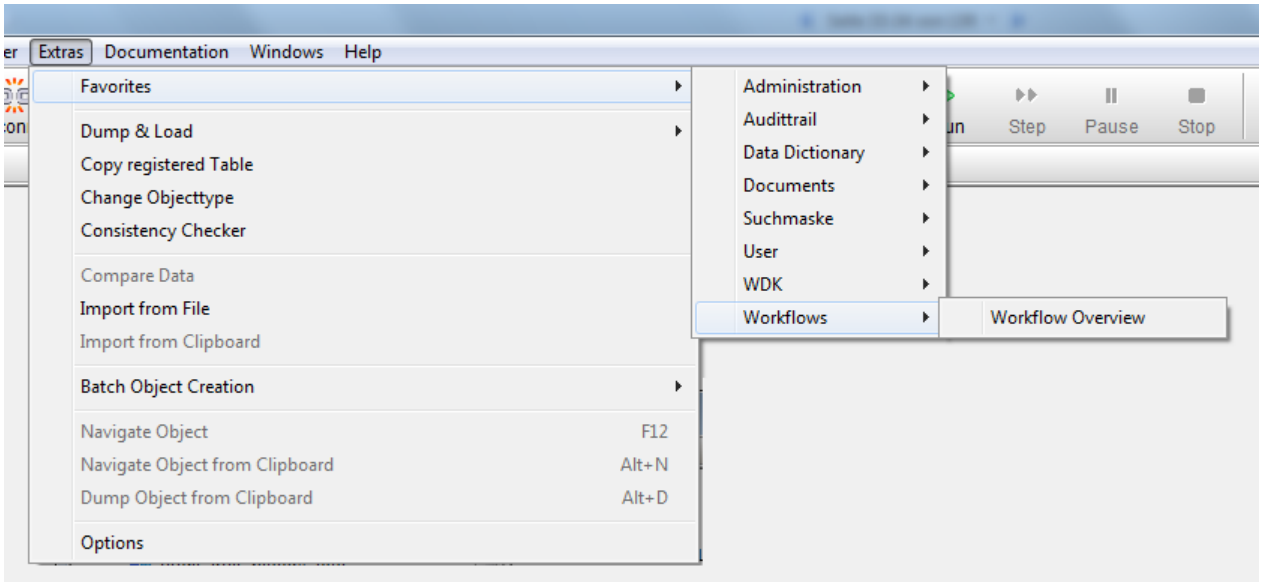


Figure 9 – Favorite menu

To open the Favorite menu, select the item *Favorites* from the *Extras* menu or click on the *Favorites* button of the toolbar.

All queries in this menu are grouped by the entered categories that in each case represent a submenu. From there the appropriate queries can be selected by their names. Selected queries are copied to the DQL command input area.

If <Ctrl> is pressed while selecting a query, the query will be inserted at the current cursor position. (Please note that template field options will not be merged automatically). Otherwise, the DQL command input area will be cleared before inserting the query.

3.1.11. Template Queries

Template Queries are DQL queries which contain placeholders for values. If a Template Query is executed, the user is requested to enter values for the placeholders.

```
select * from dm_document
  where object_name like '%{object_name}%' or
        owner_name = '{Owner}'
```

Figure 10 - Template Query

Placeholders consist of a freely selectable name, which is enclosed by curved clips. They can be inserted either manually or with a function available in the context menu of the input window. Template Queries can contain any number of placeholders.

If placeholders appear in a query more than once, they always represent the same value. Please keep in mind that placeholder names are case sensitive.

If for any reason an opening curved clip is needed in the DQL query for another purpose than for beginning a placeholder, it must be masked with a double opening curved clip, like … **`where object_name like '{{%'`**.

The behaviour of placeholders can be improved by adding placeholder options, using the *Edit template field* function from the context menu.

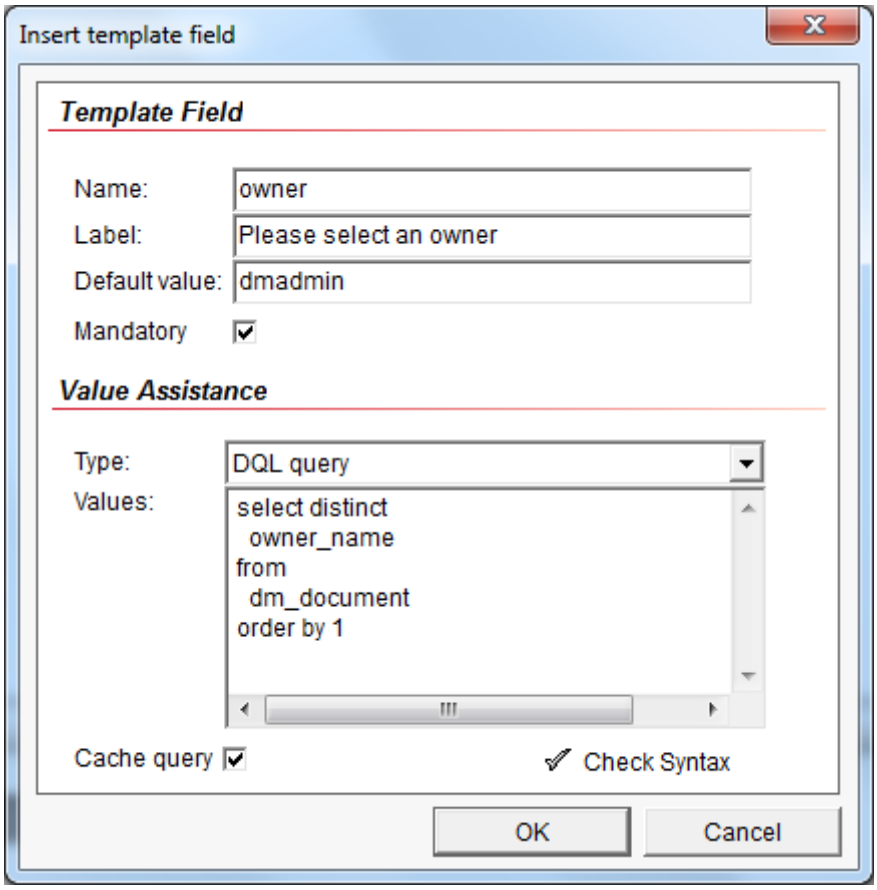


Figure 11 - Template Field options

Name:	Name of the placeholder
Label:	Caption displayed in the input dialog. If no label is available, the name will be displayed.
Default Value:	The default value shown in the input control. It is valid until another value is entered or selected.
Mandatory:	Check this box, if it is a mandatory field.
Use cached query:	If checked, the result of the DQL query used to fill the value assistance will be cached to improve performance, but changes to the underlying values will be ignored. Select *Refresh Value lists* from the *Session* menu to clear the cache.

Value List Type: To assist the user, different types of value lists are available:

None:	No value list.
Fixed List:	A list with fixed values that must be predefined in the control *Values*.
DQL Query:	A DQL query that fills the list dynamically
Date Picker:	Displays a date picker control
File Open Dialog:	Opens a file open dialog. A list of available file types can be defined in the control *Values*.
Directory Selector:	Allows selection of a directory from the local file system.
Cabinet/Folder Selector:	Allows selection of a cabinet or folder from the currently connected repository.

If the result of a DQL query contains more than one attribute, the value of the first column will be displayed in the value list, the corresponding value of the second column will be inserted in the query.

In a fixed value list, the displayed value and the inserted value can be separated by ||, (e.g. Label||Value).

The query

```
Select object_name, r_object_id from dm_document
```

will fill the list with the `object_name` attribute of all dm_document objects. Selecting a name from the list returns the `r_object_id` for the placeholder value.

Check Syntax: Check the syntax of the current DQL query. This feature is only available, if a valid session exists.

3.1.12. Formating DQL Queries

DQL queries can be formatted automatically. The associated format templates are defined in the file *dqManFormatTemplates.xml*.

To format a query, click on the query in the DQL input area and select the function *Auto Format Query* from the context menu of the DQL input area.

The Query

```
Select r_object_id, object_name, acl_name from dm_document
where object_name like 'dm_%' or object_name like 'xy_%' order by
r_object_name, acl_name
```

will be formatted to

```
SELECT
  r_object_id,
  object_name,
  acl_name
FROM
  dm_document
WHERE
  object_name like 'dm_%' or
  object_name like 'xy_%'
ORDER BY
  object_name, acl_name
```

Queries can also be unformatted with the function *Unformat Query* from the context menu. Calling this function removes all tabs, multiple spaces, line feeds and carriage returns from the query, leaving a plain single line query.

The third formatting function *Extract Query* can be used to extract the DQL part out of a piece of program code. From the code

```
String query = "Select * from pl_produkt (all) where i_chronicle_id = '" +
               chronicle_id + "' and " +
               "valid_from <= Date (Today) and " +
               "Dateadd(day,1,valid_to) > Date(Today) and " +
               "any r_version_label = '_NEW_'";
```

the query

```
Select * from pl_produkt (all) where i_chronicle_id = '' and valid_from <=
Date (Today) and Dateadd(day,1,valid_to) > Date(Today) and any
r_version_label = '_NEW_';
```

will be extracted.

Note that the variable `chronicle_id` is removed, leaving an empty value to be filled `…where i_chronicle_id = '' and…` .

3.1.13. Repeated Queries

DQL queries can be repeated automatically in a selectable interval. The function is available in the context menu of the DQL input area. (*Repeat Query every …)*

After selecting an interval, the background of the input area turns grey. Automatic execution is stopped either when the query is changed or *Stop Repeat Query* from the context menu is selected.

In case of automatic execution, the message area will not be deleted but contains a log of all executions.

3.2. The API View

Single API Commands can be executed in the API View.

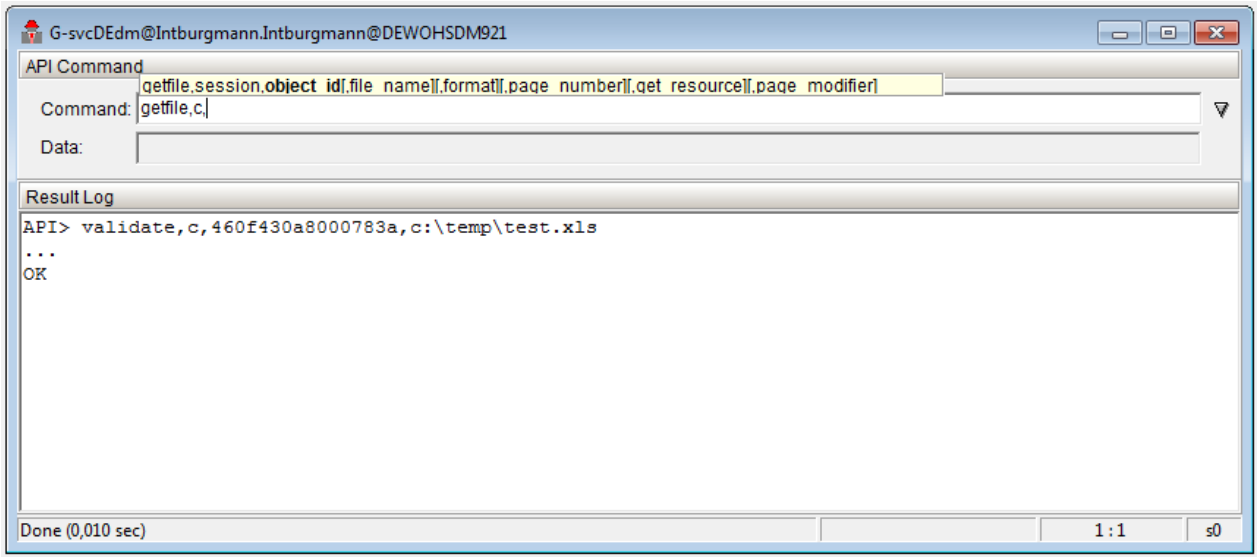


Figure 12 - The API View

To execute an API command press <Enter>, <F9> or <Shift>+<Enter>, select *Run* from the Edit menu or click on the *Run* button in the toolbar.

Double clicking an element in the result log will copy the value into the clipboard. If the <Ctrl> key is pressed while double clicking, the value will be pasted into the active input control at the current cursor position immediately.

API commands like connect, getservermap etc. that do not need an active session can be executed locally.

To open the API command help, click the button on the right. In the help window you can search for API commands, or insert a command by double click to the command input field.

If one object is selected in the DQL result list, its `r_object_id` can be copied into the command input control, pressing <Alt>+<O> or calling the function *Copy r_object_id* from the context menu.

The result log also offers a context menu with some special functions:

Dump Object	Creates a dump of an r_object_id
Open File	Opens the file, if a full filename is supplied in the log, e.g. after a getfile command.
Login as …	opens a new session with the login ticket data under the cursor *after a getlogin command.*

3.3. The DQL and API Script View

The two views DQL Script and API Script are described together, since both offer nearly the same functionality. The script is entered in the upper input area. In the lower part the results of the script are logged.

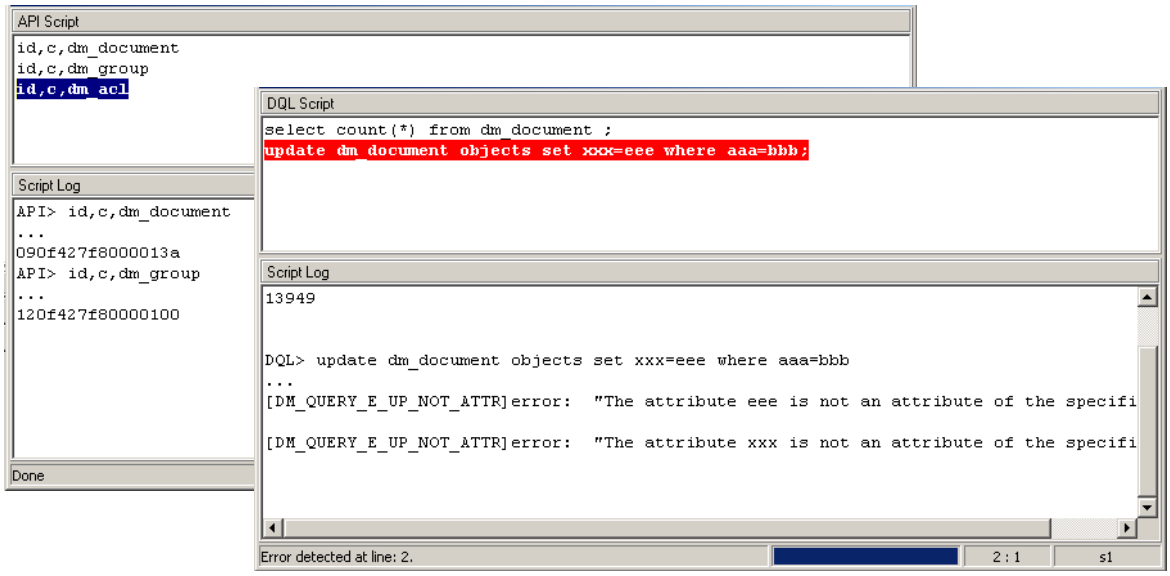


Figure 13 - API and DQL Script View

Double clicking an element in the script log will copy the value into the clipboard. If the <Ctrl> key is pressed while double clicking, the value will be pasted into the input area at the current cursor position immediately.

Scripts can be created or loaded in different ways:

- Load from file: Either with the *Load Script* function from the Edit menu, or by dropping a file on the input area.
- Created from DQL query results or imported data, using the Script Generator
- Manually typing a script

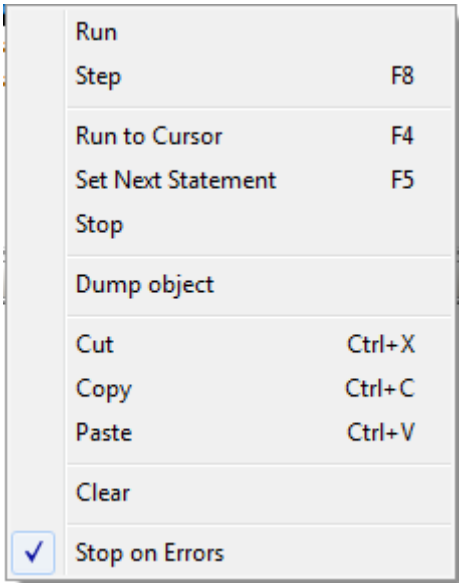
API scripts will be executed line by line.

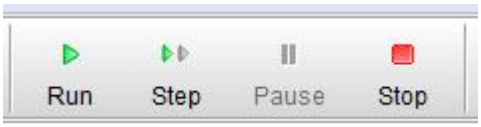
DQL commands can consist of more than one line. Therefore, commands in DQL scripts must be separated either by a semicolon at the end of each command or by an extra line containing only the command *go*.

Each line beginning with the # character or with // is regarded as comment line.

3.3.1. Executing Scripts

Scripts can be executed either as a whole or step by step. The execution can be controlled by the tool bar or the context menu.





Run: Start the script

Step: Execute the next single command. This command is highlighted. If the script has not been started, Step turns on the debug mode and selects the first command.

Pause**:** Pause a running script and switch to the debug mode. The next command will be highlighted.

Stop: Stop the execution.

Additional functions are available in the context menu.

Run to Cursor executes all commands from the current debug position or from the beginning to the currently selected line. The command in the current line is highlighted and the debug mode is switched on.

In debug mode, with *Set Next Statement* the next command to execute can be selected.

.

3.3.2. The Script Navigator

The Script Navigator is a directory tree for all available local drives, enabling fast access to stored API and DQL Scripts.

The Script Navigator is available in the API Script view and the DQL Script view.

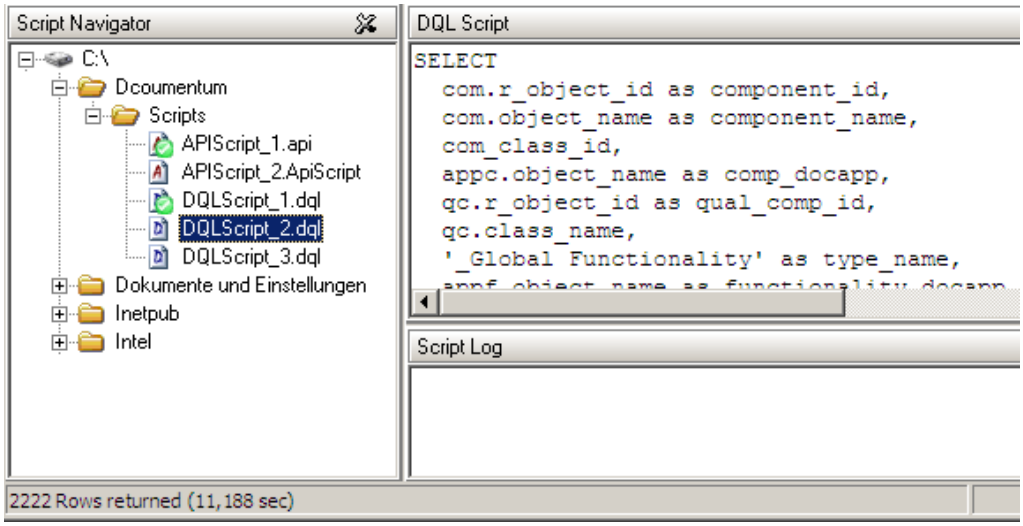


Figure 14 – Script Navigator

A selected script file can be opened using the return key or a double click. Opened scripts are marked with a special icon. These marks retain until the session window is closed. Additionally, the marks can be toggled with the function *Mark as New* and/or *Mark as Opened* from the context menu.

Default settings for file extensions are .api, .apiscript, .dql and .dqlscript. They can be configured by calling the function *Filter* from the context menu.

3.3.3. Error Handling

The option **Stop on Errors** defines the error handling behaviour. If the option is active, the script will be paused in case of any error. The inaccurate command will be highlighted. The error can be corrected and the execution continued.

If this option is inactive, the script will not stop execution. Only the error message will be displayed in the result log.

Within an API script, the error handling can be switched on or off, using the script command:

```
#$ StopOnErrors on|off
```

3.3.4. Using Variables in API Scripts

In API scripts temporary variables can be used, in order to store return values for further use. The values of the variables remain during a script run, until either the execution of the script is terminated or the content of a variable is overwritten.

Using variables in API scripts is not compatible to the Documentum API language. Scripts with variables cannot be executed by other applications.

Naming conventions:

The name of a variable can be chosen freely, but it has to be enclosed with percentage signs. They are not case sensitive. Examples:

```
%1%
%File-Contents%
%this is a variable%
```

Assignments:

Constants or results of API commands can be assigned to a variable.

Examples:

```
%Repository_Owner% = dmadmin
```

initializes the variable `%Repository_Owner%` with the value "dmadmin".

(The `Define` command from earlier dqMan versions can still be used, but is marked deprecated)

```
%ID% = id,c,dm_document where object_name like 'a%'
```

assigns the result of the API command to the variable %ID%

Concatenating values:

Values can be concatenated in two ways:

```
%ID% =+ Add this text
%ID% = %ID%  Add this text
```

Usage:

To use a variable, insert it at the appropriate position in the API command:

```
get,c,%ID%,object_name
```

Query variable contents

While executing the script in debug mode, values of variables can be queried at any time. Click on the variable and wait for the hint to appear.

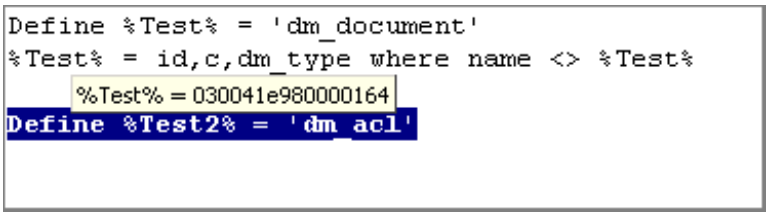


Figure 15 – Querying Variable Contents

3.3.5. Using Script Commands in API Scripts

Script commands are additional commands that expand the functionality of API scripts or object functions. Using script commands in API scripts is not compatible to the Documentum API language. Scripts with script commands cannot be executed by other applications.

Available script commands are

`#$StopOnErrors on|off`	Sets the error handling behaviour during script execution.
`#$Print <text>`	Prints the text and/or the values of variables into the script log.
`#$Sleep <time>`	Pauses script execution for <time> milliseconds.
`#$Loop <collection identifier>`	Start of a loop through a collection.
`#$Loop <attribute_name(object_id)>`	Start of a loop through the values of the repeating attribute from the object identified by the ID object_id.
`#$EndLoop`	Marks the end of the loop.

Example for a collection loop:

```
%Collection%=readquery,c,select group_name from dm_group where any
i_all_users_names = 'dmadmin'

#$Loop %Collection%
  %group_name% = get,c,%Collection%,group_name
  #$ Print Group name %LoopIndex1%: %group_name%
#$EndLoop
```

The handling of the collection is done by the `#$EndLoop` command. It is not necessary to add next or close commands.

The variable `%LoopIndex<i>%` does only exist within a loop and contains the current loop index, starting with 0. `%Loopindex1%` is the index of the first loop. `%Loopindex2%` would be the index of the second loop, if it is nested within the first loop.

Example for an attribute loop

```
%id%= id,c,dm_document

#$Loop i_folder_id(%id%)
  %Folder_ID% =get,c,%id%,i_folder_id[%LoopIndex1%]
  #$Loop r_folder_path(%Folder_ID%)
    %FolderPath%=get,c,%Folder_ID%,r_folder_path[%LoopIndex2%]
    #print result
    #$ Print %FolderPath%
  #$EndLoop
#$EndLoop
```

The variable `%LoopIndex<i>%` is incremented each time the `#$EndLoop` command is executed. It can be used as index for the repeating attribute values.

3.3.6. Logging

During the execution of scripts the result of the commands will be logged in the result log. The logging behaviour can be defined with the *Settings* option from the result log context menu.

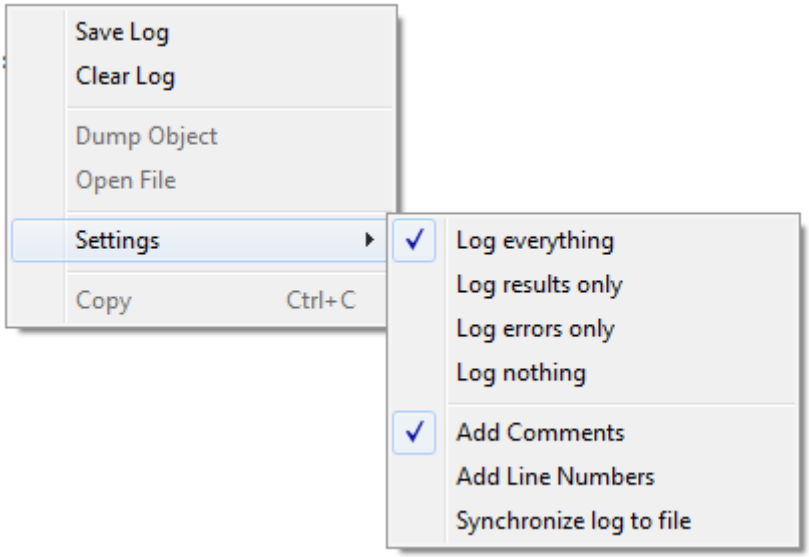


Figure 16 – Logging settings

The following logging settings are available:

Log everything:	Everything will be logged.
Log results only:	Only results or error messages will be logged.
Log errors only:	Only error messages will be logged.
Log nothing:	Nothing will be logged.
Add comments:	Comments from the script will be copied into the log.
Add line numbers:	The line number of the command will be added to the script. If line numbers are stored, the function **Show command** allows jumping from any line in the log to the corresponding command in the script.
Synchronize log to file:	This options allows copying all log messages into a file. If this option is selected, the user has to select a target file every time a script is started. If an existing file is selected, the additional content will be appended to the existing.

CSV format for Query results: (*Only valid for DQL Script Log)*

Results of DQL queries will be logged in CSV format.

Fixed width for Query results: (*Only valid for DQL Script Log)*

Results of DQL queries will be logged as list with fixed width. Values can be truncated, if too long.

4. Creating a Dump of an Object

From almost every input and output field a dump can be provided, if a valid `r_object_id` is available. The „*Dump object*" function can be called from the appropriate context menu, if it is opened on the `r_object_id`. The dump will be displayed in a separate window.

Dump object from Clipboard

If the clipboard contains a valid `r_object_id`, a dump can be generated directly with the function „*Dump from Clipboard*" in the *Extras* menu. The function can also be called using the shortcut <Alt> + <D>.

If a valid session cannot be found, first the login history will be searched and in the case of success, an automatic login will be established. If no session can be found or the automatic login fails, the login dialog will be displayed.

All created sessions will be closed immediately after generating the dump.

5. Search

All text areas, tables and the DQL history can be searched using the *Search* function from the *Edit* menu. It is also available by using the shortcut <Ctrl> + <F3>.

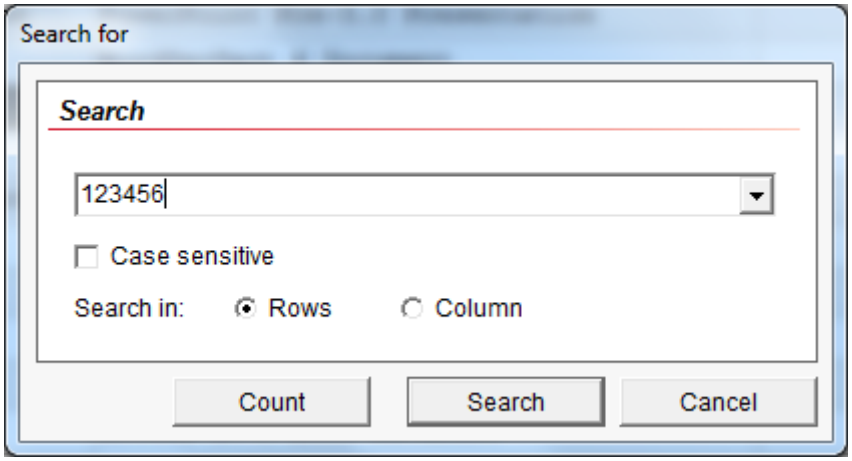


Figure 17 - Search

Enter your search term into the *Search for* field. A history is available through the dropdown list.

If a table is searched, the order of search directions can be determined, either by rows first or columns first.

To count all occurrences, click on the *Count* button.

6. The History

The *History* records all successfully executed DQL and API commands for further use. It is controlled using the tool bar.

Figure 18 – History controls

Clicking on *Back* or *Next* will load the according query into the input control, replacing the existing content.

The *History* list shows a list of all recorded commands..

To copy a command either double click it or use drag and drop. The content of the target input area will be replaced, unless a part of it has been selected.

Copying a command to a script view will insert the command at the current cursor position without overwriting the existing content, unless something is selected.

7. The Repository Navigator

The Repository Navigator is available in the DQL view. It can be opened or closed by the *Repository Navigator* item from the *View* menu, or by the *Navigator* button of the toolbar.

The Repository Navigator offers a cabinet/folder tree, an object type/registered table tree, a group tree and a favorites tree for the currently connected repository of the session window.

Every time a tree item is expanded, the appropriate data is fetched dynamically from the repository. Therefore, an update of a part of the tree can be easily done by collapsing and re-expanding the parent folder.

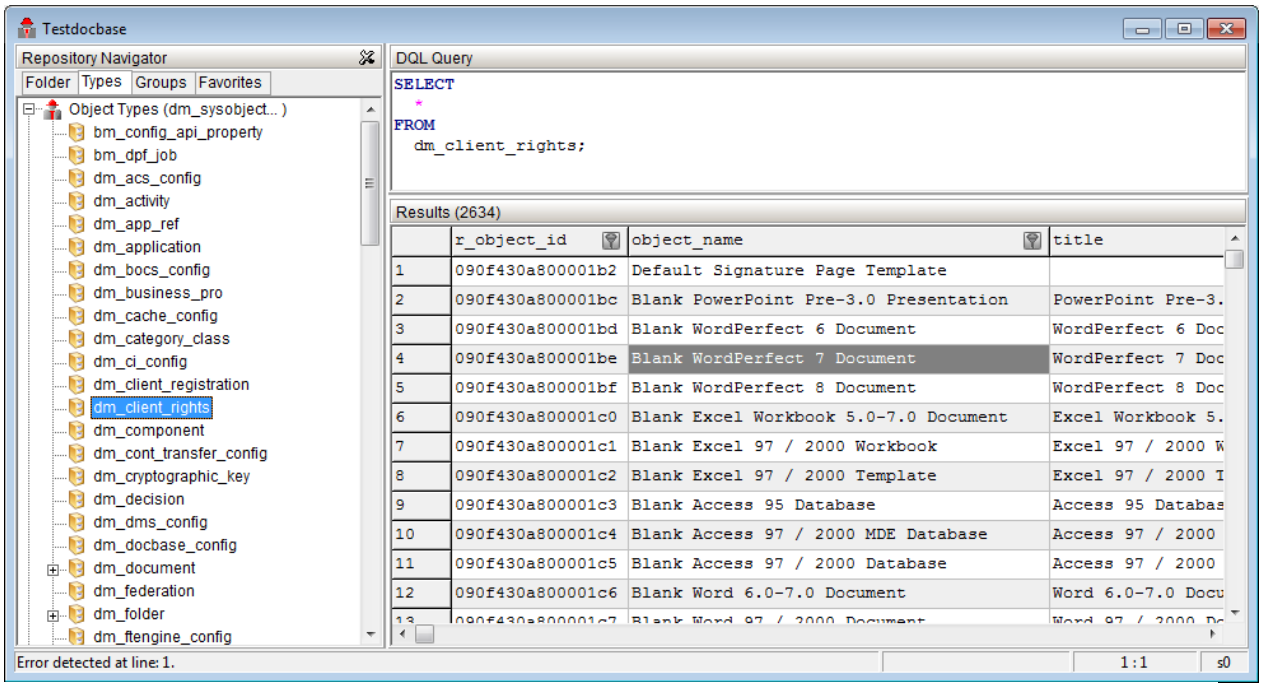


Figure 19 –Repository Navigator

7.1. The Cabinet/Folder Tree

Creation of a DQL query for the folder contents
Clicking on a folder with the left mouse button directs dqMan to create a DQL query for the folder contents. It is placed in the DQL command input area without executing it. The query is created by using the *Default Query Template* that can be modified in the options dialog.

Viewing the folder content
Using the center mouse button creates a DQL as described above, but it is automatically executed.

Context menu
Clicking on a folder with the right mouse button opens a context menu:

- Expand recursive: Expands all subfolders.
- Dump object: Creates a dump of the currently selected folder.
- List of query templates: List of template queries, defined in the options dialog. Templates marked with a green arrow will be executed immediately; the others generate a DQL query only.
- Create Folder: Opens a dialog to create a folder. The folder is linked into the currently selected cabinet or folder. The name of the folder is mandatory; an ACL can be assigned optionally.
- Create Document: Opens a dialog to create a document (dm_document or any of its subtypes). The document is linked into the currently selected cabinet or folder.
  The selection of the object type and the object name are mandatory. Assigning a content file from the local file system or an ACL is optional.
- Destroy Cabinet: Recursively destroys all subfolders of the currently selected folder, including all contained objects. If these objects are linked to more than one folder, only the links can be destroyed by user choice.
- Tree Options Opens the appropriate page of the options dialog to manage repository tree settings.

7.2. The Object Type/Registered Table Tree

Creation of a DQL query for type objects or registered tables
Clicking on an object type/table with the left mouse button directs dqMan to create a DQL query for objects of the selected type/rows of the selected registered table using the *Default Query Template*. It is placed in the DQL command input area without executing it.

Viewing the objects
Using the center mouse button creates a DQL as described above, but it is automatically executed.

Context menu
Clicking on an object type/table with the right mouse button opens a context menu:

- Expand recursive: Expands all subtypes.
- Dump type object: Creates a dump of the currently selected type.
- List of query templates: List of template queries, defined in the options dialog. Templates marked with a green arrow will be immediately executed. The others generate a DQL query only and place it in the DQL command area.
- User Functions List of user functions available for the selected tree item.
- Tree Options Opens the appropriate page of the options dialog to manage repository tree settings.
- Copy Registered Table Opens the Copy Registered Table assistant and preselects the table.

7.3. The Group Tree

Creation of a DQL query for group objects
Clicking on a group with the left mouse button directs dqMan to create a DQL query for the group using the *Default Query Template*. It is placed in the DQL command input area without executing it.

Viewing the type objects
Using the center mouse button creates a DQL as described above, but it is automatically executed.

Context menu
Clicking on a group with the right mouse button opens a context menu:

- Expand recursive: Expands all subgroups. As the group structure of a repository is not organized hierarchically, it is possible that groups do contain each other. In this case this function will result in an endless loop.
- Dump group object: Creates a dump of the currently selected group.
- List of query templates: List of template queries, defined in the options dialog. Templates marked with a green arrow will be executed immediately; the others generate a DQL query only.
- User Functions List of user functions available for the selected tree item.
- Tree Options Opens the appropriate page of the options dialog to manage repository tree settings.

7.4. The Favorites Tree

Load a favorite query
Clicking on a favorite with the left mouse button loads the DQL query of the favorite into the DQL command input area without executing it.

Run a favorite query
Using the center mouse button loads the DQL query as described above, but it is automatically executed.

Context menu
Clicking on a favorite with the right mouse button opens a context menu:

- Run Favorite: Loads the DQL query of the selected favorite and executes it immediately.
- Manage Favorites Opens the manage favorites Dialog, as described in section *3.1.9 DQL* Favorites*.*
- Expand all: Expands all subfolders.

8. The Object Navigator

The intention of the Object Navigator is to provide an easy, fast and comprehensive access to the properties and connections (like ACLs, owner names, relations, subscriptions, lifecycles, workflows …) to any other object in the repository.

All properties can be edited directly in the Object Navigator. Additionally, you can follow connections to other objects similar to a web browser by simply clicking on the link. The dependent object will then be opened in the Object Navigator.

Furthermore, all available actions like object functions, type scripts and plugins are also available making Object Navigator first choice of working with single objects.

There are different ways to open Object Navigator or to load new objects.

- From anywhere within dqMan, an explicitly identifiable object can be used. These are r_object_ids in any text area like lists, logs, tails etc., any object in the Repository Navigator, or distinct values in the DQL result list like acl_name, owner_name, r_object_type etc..
  After selecting an identifier, the object can be loaded into Object Navigator by either pressing <F12>, selecting *Navigate Object* from the *Extras* menu or by executing a delayed mouse click (press the left mouse button on the object for about one second, until mouse cursor changes. Then release mouse button).

- If the clipboard contains a valid r_object_id, this object can be loaded directly by pressing <Alt> + <N> or selecting *Navigate Object from Clipboard* from the *Extras* menu.

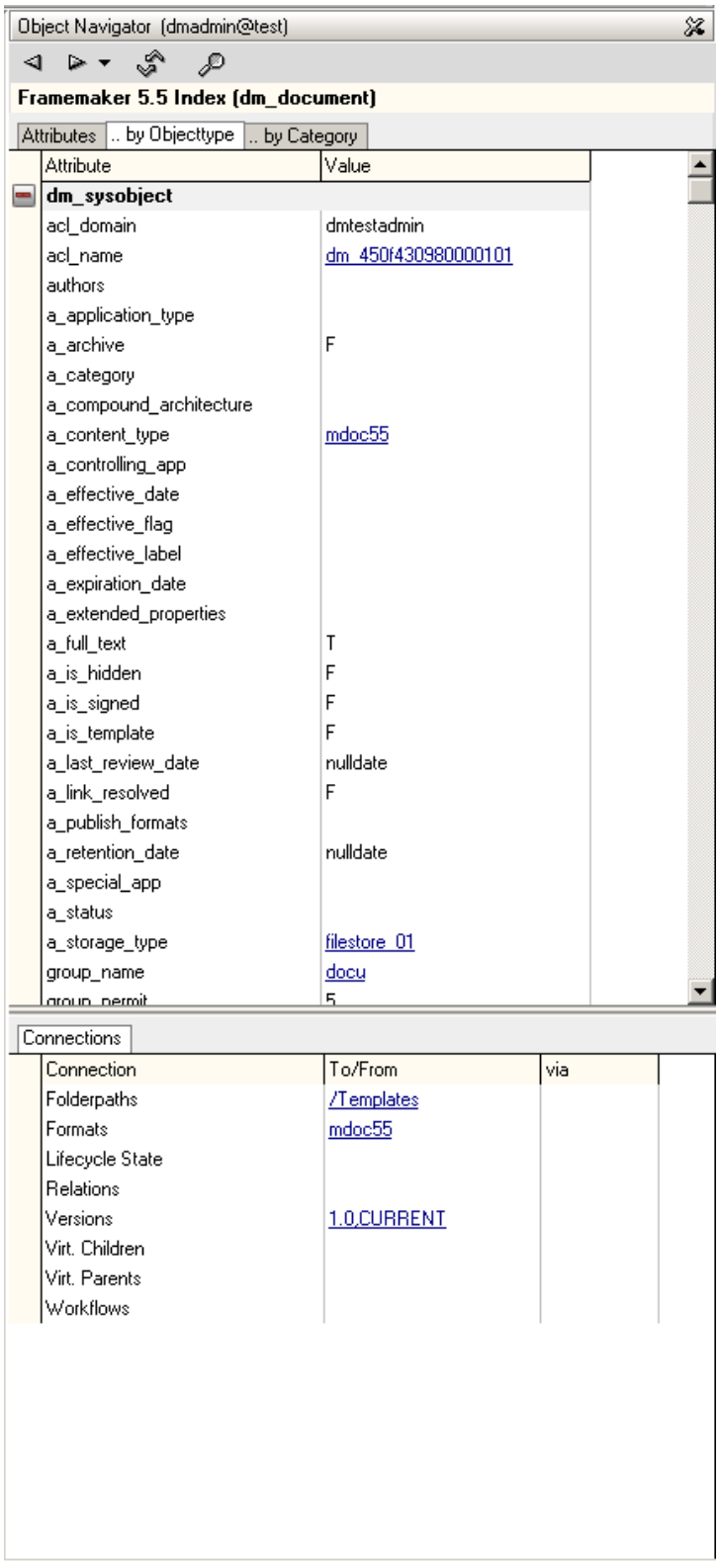
- Follow a link to a dependent object as described above.

- Use <Alt> + <O> or select *Object Navigator* from the *View* menu to open it without selecting any object. This is useful to show an already loaded object again or to use the *Quick Pick* function (see below) to directly search for an object.

- Load an object within an object function by using the script command *FetchObject*

In any case the Object Navigator will open if it is not already visible.

The Object Navigator is divided in different areas:



Navigation Bar
Title Bar

Attribute Area

Connections Area

The navigation bar offers some functions for handling objects:

History of loaded objects

Reload button to refresh currently displayed object

Quick Pick, a fast search for objects

The Quick Pick search function offers fast access to objects in the repository. The top edit field expects 2 strings to begin a search.

The first is part of the object type to be searched in. You can either enter the complete type name, like dm_user, or part of the type name, e.g. user. In the latter case, the search will be conducted in all types containing the keyword “user” in the type name (dm_user, dm_message_user_data, dm_message_route_user_data, dmc_workqueue_user_profile and dmc_wq_user_skill).

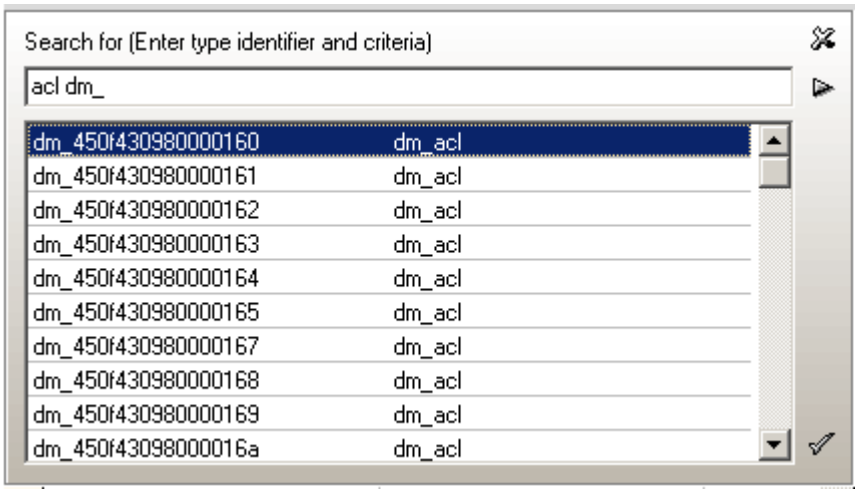
The second string is a part of the object name.

Valid searches are for example

- user admin
- acl preset
- folder template
- ….

Press return or click on to start the search. Search results are displayed in a list and can be selected either by double clicking on it, selecting it with up and down arrow keys and pressing enter or by a click on . The selected object will be opened in the Object Navigator.

The result list will be available until the next search is started.



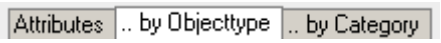
If only one string is provided, dm_sysobject is assumed to be the type to search in.

In the title bar, the name of the currently selected object and its type are displayed.

The attribute area displays all attributes of the object and their values which can be edited within the navigator. The context menu of the attribute area offers more functionality:

Copy	Copies the selected value to the clipboard.
Copy value of	Shortcut to copy selected value of the object to the clipboard, please see section Configuration for more info.
Edit	Opens an inline editor to edit the selected property value.
Add to compare List	Adds the current displayed object to the *Object Compare List*
Destroy Cabinet/ Destroy Folder	Recursively destroys all subfolders of the currently displayed folder and the displayed folder itself, including all contained objects. If these objects are linked to more than one folder, only the links can be destroyed by user choice.
Create ticketed user session	Creates a dqMan session for the displayed user by generating a login ticket.
Object Functions	This section lists all object functions available for the current object.
Type Scripts	Available type scripts for the current object.
Plugins	Plugin functions for the current object.

The tabs on top of the attribute area offer different ways to order the attributes.



Attributes	alphabetical order
by Objecttype	ordered by type hierarchy
by Category	classical dump order by category

The **connection area** contains more links to other objects, depending on the object type. If this connection is using another object, like a dm_relation object, the link to this object is also provided in the column “via”.

Predefined connections are for example:
Relations, subscriptions, ownerships, memberships, virtual parents or children, lifecycles, workflows, folder links, etc.

The listed connections are usually limited to 50 for performance reasons. If all are needed, the context menu offers a function to execute the underlying query in the DQL window.

Configuration

The Object Navigator configuration is stored in the file *dqManObjectNavigator.xml* in *the /config* subdirectory of the dqMan home directory (c:\program files\dqMan).

The configuration is done in different sections within the file:

Section `<identifiers>`

Identifiers are used to uniquely identify objects by the value of an attribute (e.g. an acl_name identifies a dm_acl object) and are used in the DQL result list or the Object Navigator itself.

The attributes can be single or repeating. Identifiers consist of an attribute name (`columnname="acl_name"`) and a qualification (`qualification="dm_acl where object_name='{*}'"`). The value of the attribute is represented by {*}.

```
<identifier columnname="acl_name"
            qualification="dm_acl where object_name='{*}'" />
<identifier columnname="a_content_type"
            qualification="dm_format where name='{*}'" />
<identifier columnname="content_type"
            qualification="dm_format where name='{*}'" />
<identifier columnname="r_accessor_name"
            qualification="dm_group where group_name='{*}' union
            select r_object_id from dm_user where user_name='{*}'
            and r_is_group=0" />
```

Section <copy_attributes>

Copy attributes fill the submenu of the function *Copy Value of* from the attributes area. It is a simple list of all possible values. The submenu is filled with all attribute names that are valid for the currently loaded object.

```
<copy_attributes>
  <attribute name="r_object_id" />
  <attribute name="object_name" />
  <attribute name="group_name" />
  <attribute name="user_name" />
</copy_attributes>
```

Section <translations>

Translations are used to make cryptic attribute values more readable. (e.g. Boolean values can be translated to true or false). To take effect, translations have to be assigned to attribute values in the objecttypes section.

```
<translation name="user:user_state" bitwise="F">
  <value source="0">Active</value>
  <value source="1">Inactive</value>
</translation>

<translation name="boolean" bitwise="F">
  <value source="T">True</value>
  <value source="F">False</value>
</translation>
```

Section <objecttypes>

The objecttype section assigns translations to the attributes of an object type and defines queries for connections to other objects. Object types can inherit definitions from other object types.

The type dm_cabinet inherits all settings (translations and connections) from dm_folder. Additionally the attribute “is_private” is assigned the translation named boolean:

```
<objecttype name="dm_cabinet" extends="dm_folder">
  <attributes>
    <attribute name="is_private" translation="boolean" />
  </attributes>
</objecttype>
```

Connections are defined as a DQL query, whereas defined labels are used to assign the result to the Object Navigator columns.

```
<relation name="packages" label="Workflow Packages" limit_to="50">
  select
    r_object_id as relation_id,
    'dmi_package' as relation_label,
    r_component_id as target_id,
    r_component_name as target_label
  from dmi_package
  where r_workflow_id = '{r_object_id}'
  order by 4
</relation>
```

The number of results of a query can be limited by setting the `limit_to` value for the connection. This is useful to limit loading time of an object.

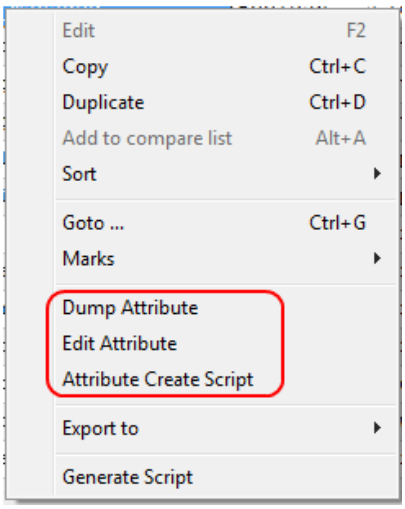
Attributes in curved brackets reference the currently loaded object in the navigator.

The resulting values are displayed in the Object Navigator: target_label in the column “To/From” and relation_label in “via”. If the label is omitted, the target_id or relation_id will be displayed instead. If no relation value is queried, the column “via” remains empty.

This chapter describes only the basic principle of the Object Navigator configuration. There are many more examples to be found in the *dqManObjectNavigator.xml.*

9. Processing Attribute Definitions

dqMan offers functions to view, edit and copy attribute definitions. They are available from the context menu of the DQL result list after executing a `describe` command.



The attribute properties always apply to the described type, not to the type the attribute is inherited from.

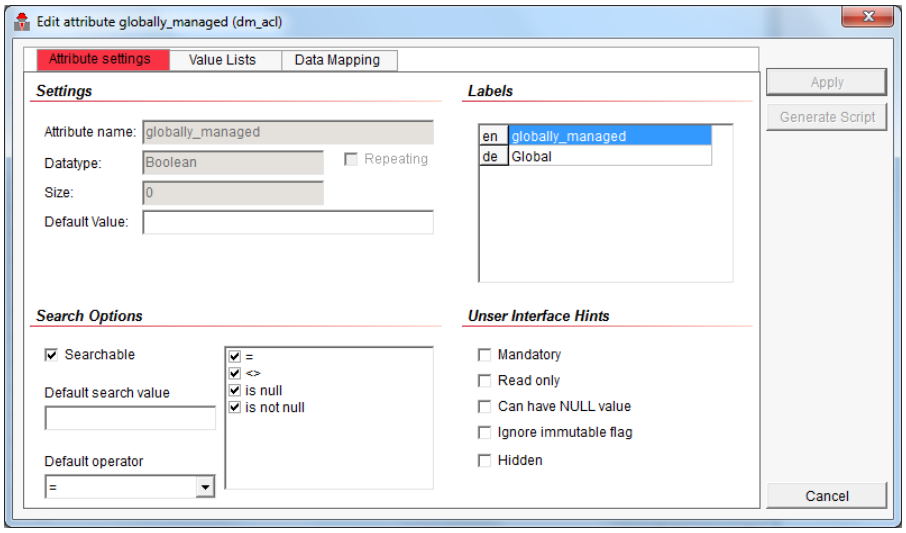
9.1. Dump Attribute

A dump of an attribute lists all definitions from the data dictionary similar to an object dump.

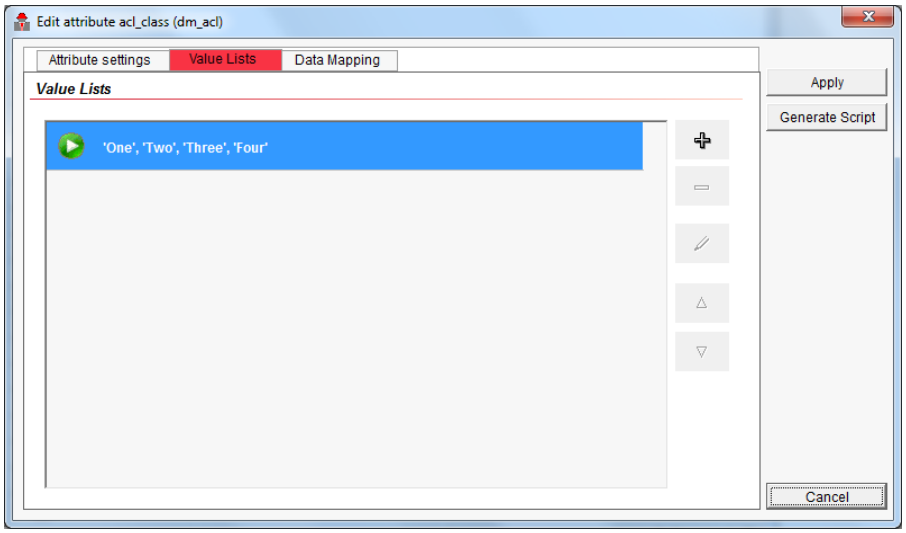
9.2. Edit Attribute

In the attribute editor, often used attribute definitions can be modified. These are general settings, search options, labels, user interface hints, value assistance and value mapping. The editor allows working with all locales without switching the session locale. All changes can either be applied directly, or a script can be generated for later or multiple use.

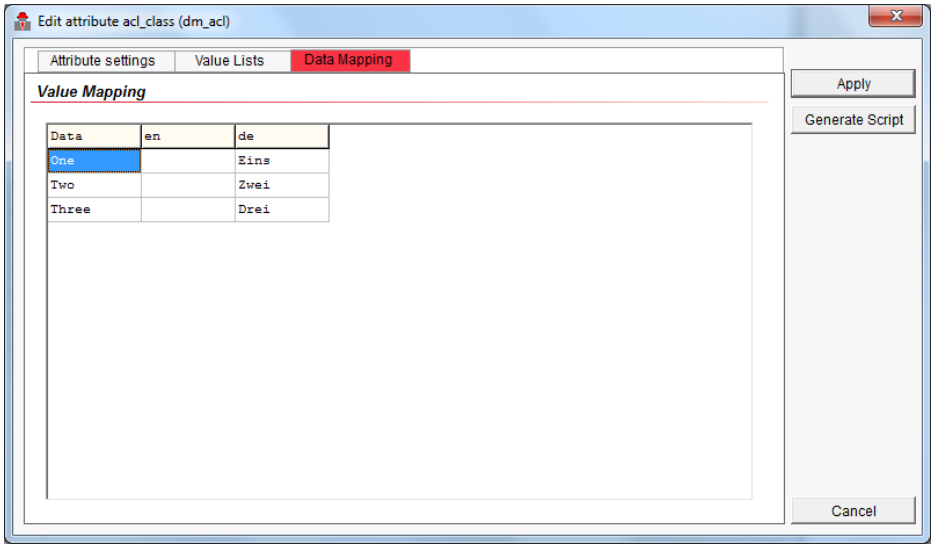
Edit attribute settings



Add, remove or reorder value lists

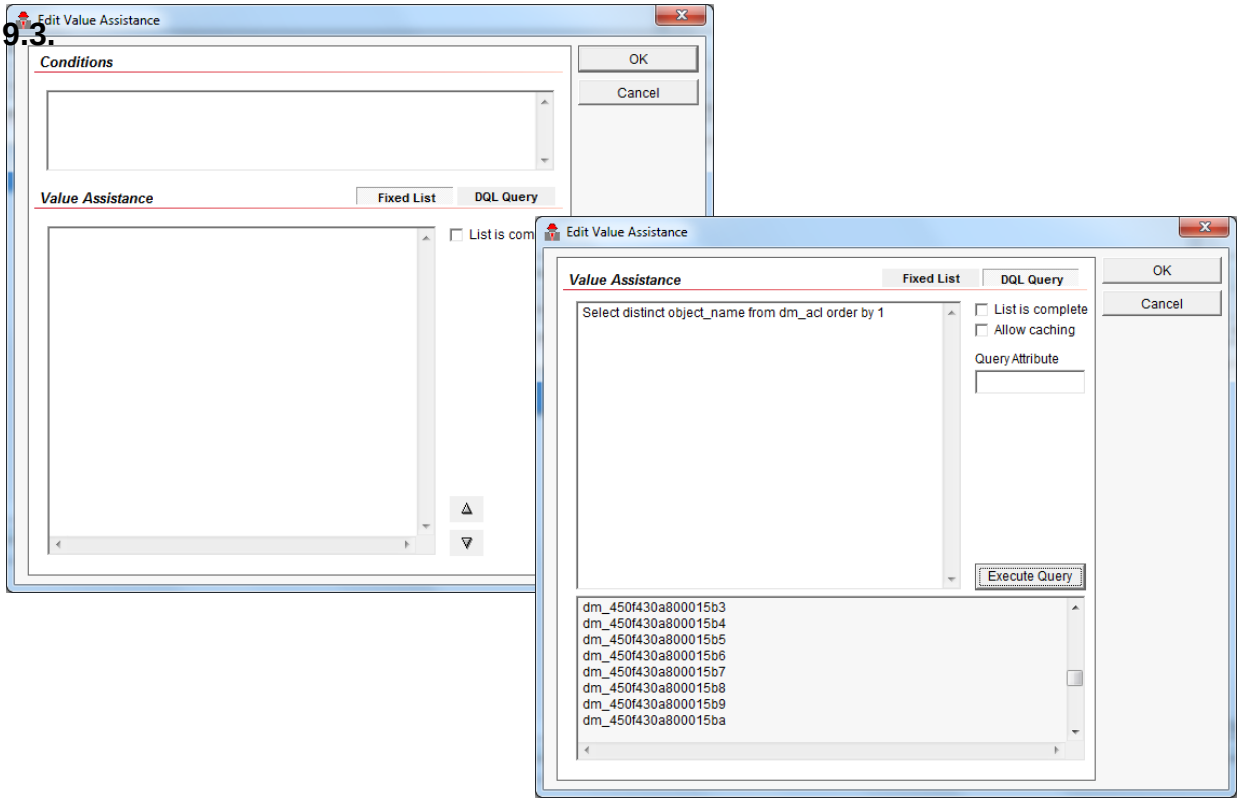


Manage value mapping, even with copy & paste of entire lists, and with all locales in one dialog



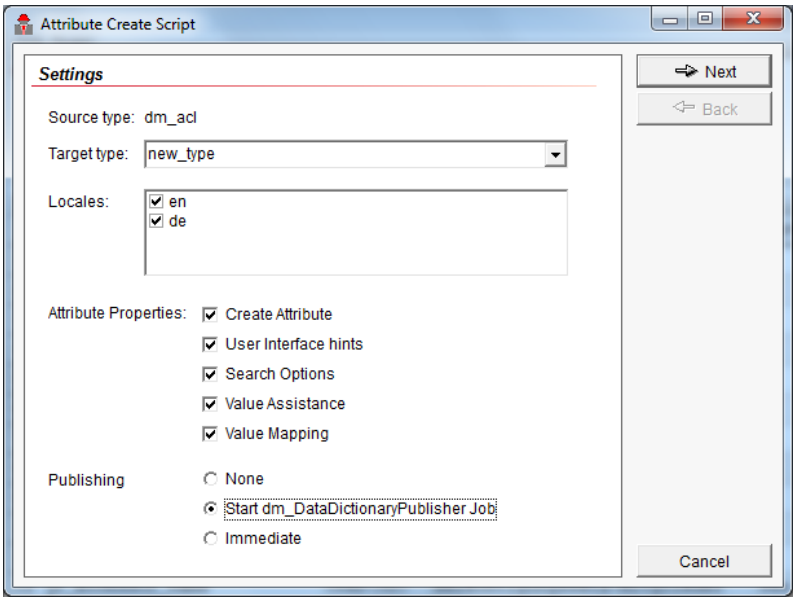
Edit value assistance content, order fixed lists or execute DQL queries to control results.

9.3.

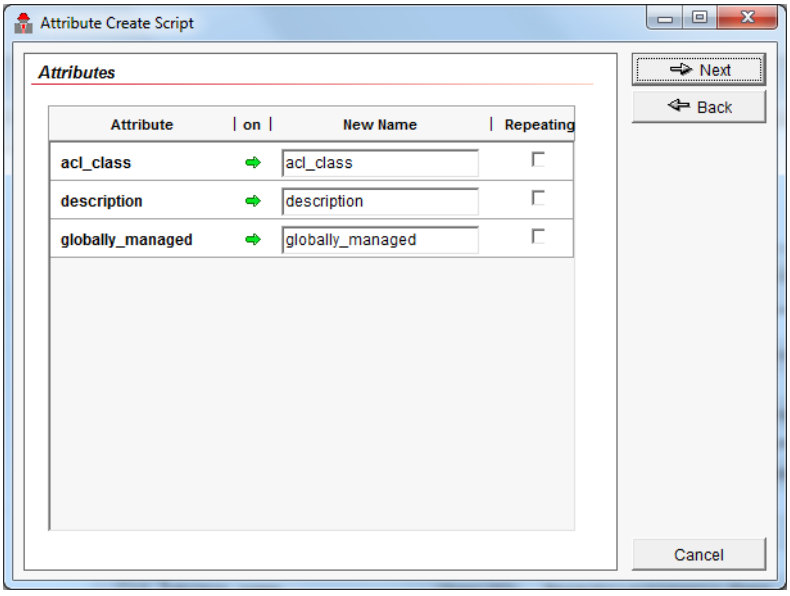


Copy Attribute

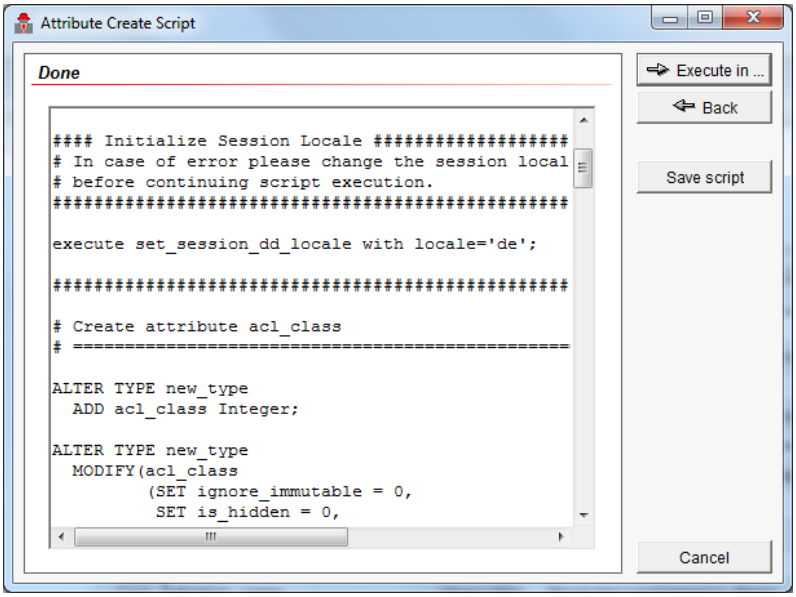
This function creates a script from existing attributes. It can be used to copy attributes between different object types or between repositories.



Select target type, locales to copy, attribute property areas and publishing options.



Rename attributes and change repeating flag if necessary.



Review generated script. Click on *Back* to change settings, *Save script* to save the script to the file system or choose a session to directly execute the generated script.

10. Viewing Textfiles

dqMan offers two functions to view external text files like log files or trace files. To open static text files, the function *View File* from the *View* menu can be used. To track changes, a tail view can be opened from the *View* menu.

In both cases, a submenu with the recently opened files is displayed. To open a new file, the item Open can be selected.

10.1. File View

This function displays the entire contents of a file. Use the context menu to access additional functions like refresh, search, dump object, or word wrap settings.

10.2. Tail View

The tail view shows a configurable number of lines from the file tail. It is refreshed regularly. The content menu offers specific functions for this viewer:

Pause/Continue:	Pauses the refresh.
Tail length (xxx):	Number of displayed lines of text.
Interval (xx seconds):	Refresh interval in seconds.
Wordwrap:	Turns on or off automatic wrapping of lines exceeding screen width.

11. The Script Generator

With this tool, DQL, API, Docbasic or other scripts can be generated from a data list, either a DQL query result list or an external data list. It is available from the context menu of the list calling the function *Generate Script*.

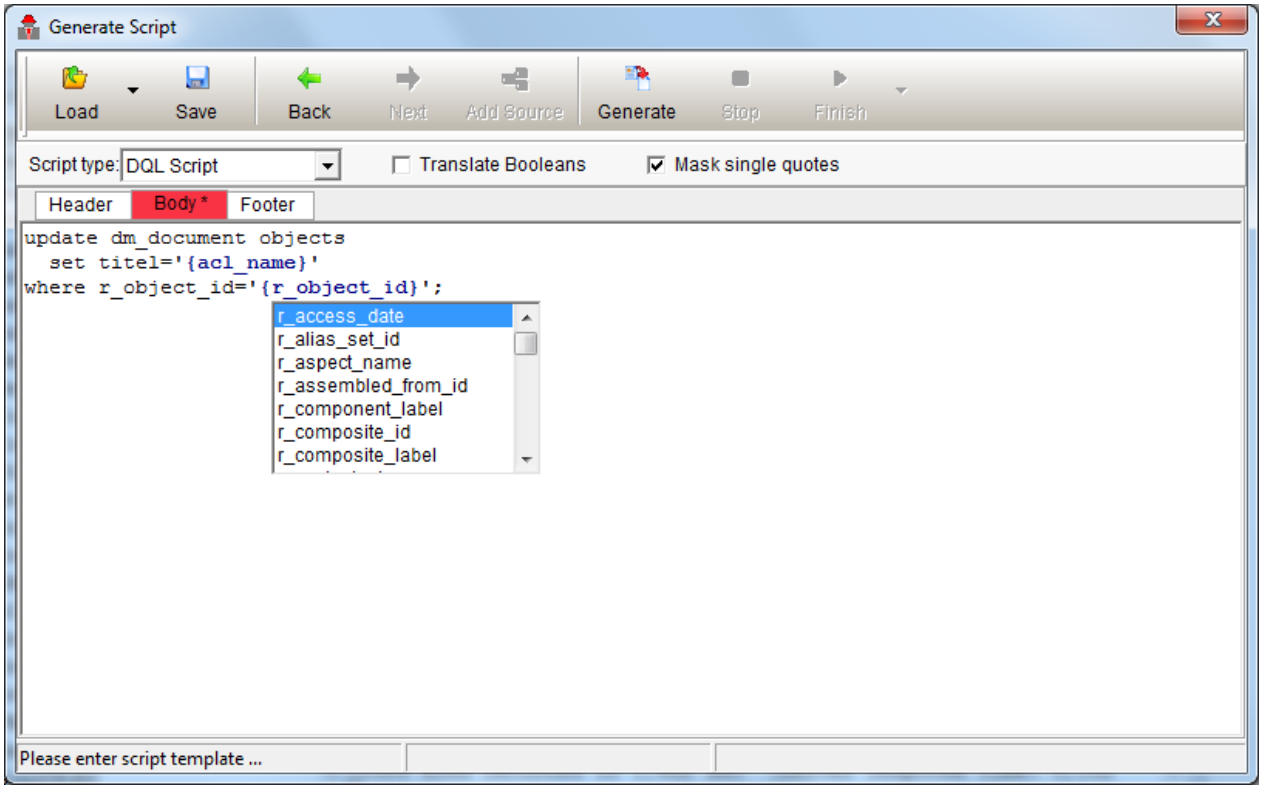


Figure 20 - Script Template Editor

11.1. Script Templates

The basis for a script is a script template. It consists of three areas: header, body and footer. Header and footer can only contain static text. It is added exactly once at the beginning and/or at the end of the Script.

The script body is applied to each line of the data list. The script contains thereby one copy of the script body for each data record. The script body can contain text, placeholders for data from the data list and processing instructions.

With the generation of the script the placeholders are replaced by data from the data list. The processing instructions are removed.

11.1.1. Placeholders

A placeholder must exist in the source data list. It consists of a column name enclosed by curved clips. Examples: `{r_object_id}` or `{object_name}`.

If the data list is a DQL query result list, placeholders can contain any attribute of the underlying object type. If this is available in the data list, the value is taken from there. If not, the value is loaded from the repository object.

Inserting placeholders can be done by clicking the *Insert Querycolumn* item in the context menu. It opens a submenu, containing all available placeholders that are part of the data list.

Entering a left curved clip in the editor area or pressing <strg>+<space> will open a select list containing all attributes of the referenced object type.

Placeholders with no reference to the data list will be displayed in italic style.

11.1.2. Processing instructions

There are two processing instructions available, LOOP and IF EMPTY.

11.1.2.1. The LOOP Instruction

LOOP allows the processing of values of repeating attributes. Example:

```
{LOOP r_version_label}
 ...{r_version_label}...
{END LOOP}
```

A repeating attributes must be specified in the LOOP instruction. Only by this the attribute will be resolved to single values. To process more than one repeating attribute simultaneously, all attributes can be specified, separated by commas:

```
{LOOP r_accessor_name, r_accessor_permit, r_accessor_xpermit}
```

This is called a combined loop. How often a loop is recurred is determined by the amount of values in the first repeating attribute. Loops can be created from the context menu.

11.1.2.2. The IF EMPTY Instruction

This instruction allows the processing of empty attributes. Example:

```
{IF EMPTY(title)}
   ...
   set,c,{r_object_id},title
   ...
{ELSE}
....
{END IF}
```

The attribute must be defined in brackets. Within a loop, repeating attributes can be processed.

IF EMPTY instructions must not be nested.

11.1.3. Examples

Example 1: Concatenate attribute values into a separate attribute.

This example will produce an API script that concatenates the values of `object_name` and `title` in the attribute `subject`, separated by a hyphen.

```
set,c,{r_object_id},subject
{object_name}-{title}
save,c,{r_object_id}
```

Example 2: Loop a repeating attribute:

This example turns all values of `authors` to capital letters. The basic DQL query (`select *, Upper(authors) as uauthors from dm_document`) supplies the values in capital letters in the repeating attribute uauthors.

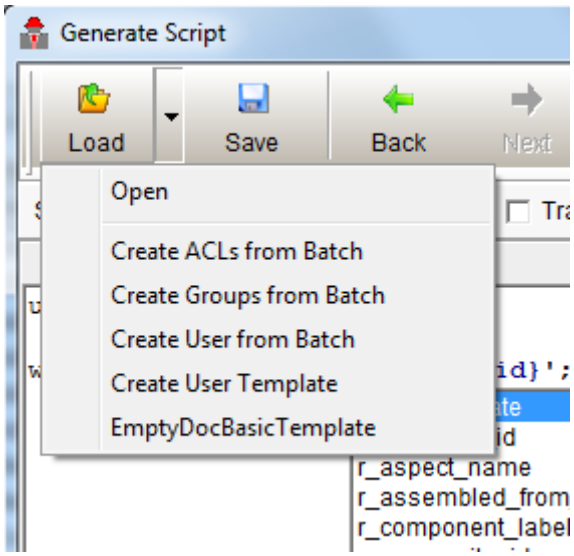
```
truncate,c,{r_object_id},authors

{LOOP uauthors}
append,c,{r_object_id},authors
{uauthors}
{END LOOP}

save,c,{r_object_id}
```

11.2. Managing Script Templates

Script templates can be saved for further use by clicking on the *Save* button or can be opened by clicking on the *Load* button of the toolbar.



The Load template button opens an Open File Dialog, where a template can be opened from the file system.

The downward arrow opens a submenu that contains all templates saved in the subdirectory ".\Scripttemplates" inside the dqMan program directory. They can be opened by a mouse click.

11.3. The History

Earlier created script templates are automatically stored in the history. To load templates from the history, use the *Back* and *Next* buttons from the toolbar.

11.4. Script Types and Options

Selecting the script type enables dqMan to enhance the script generation process.

API Script: API scripts will be pasted into the API script view for execution. A syntax help for API commands is available.

DQL Script: DQL scripts will be pasted into the DQL script view for execution. Command separation will be checked.

Docbasic Script: Docbasic scripts will be opened inside a Docbasic interpreter if available. The Docbasic interpreter and the parameters can be defined in the options dialog.

Other ...: Scripts without type cannot be executed, only saved.

Translate Booleans: Automatic translation of Boolean values. DQL queries return for Boolean values either 0 or 1. API commands expect T or F.

If this option is activated, the values will be translated, if an API script is generated. 0, T and t will be translated to T, all other values to F.

Mask single quotes This option is not available for API scripts. If a value contains a single quote, it is masked by another single quote.

For example, if a value is “Mark’s test”, a generated DQL command would contain something like … set title=’Mark’s test’… . This will cause an error on execution.

Activating this option will result in: … set title=’Mark’’s test’… that will be accepted by DQL processor and result in the correct title value.

11.5. Adding Data Sources

A script can be generated from 2 data sources simultaneously. The additional source can be another DQL query result or a data list. The additional data source can be opened clicking on the *Add Source* button of the toolbar. A submenu appears that contains all possible data lists.

The additional data source must be joined to the main data source, to allow correlation of the data records. At least one column of the main data source must be joined to one column of the additional data source

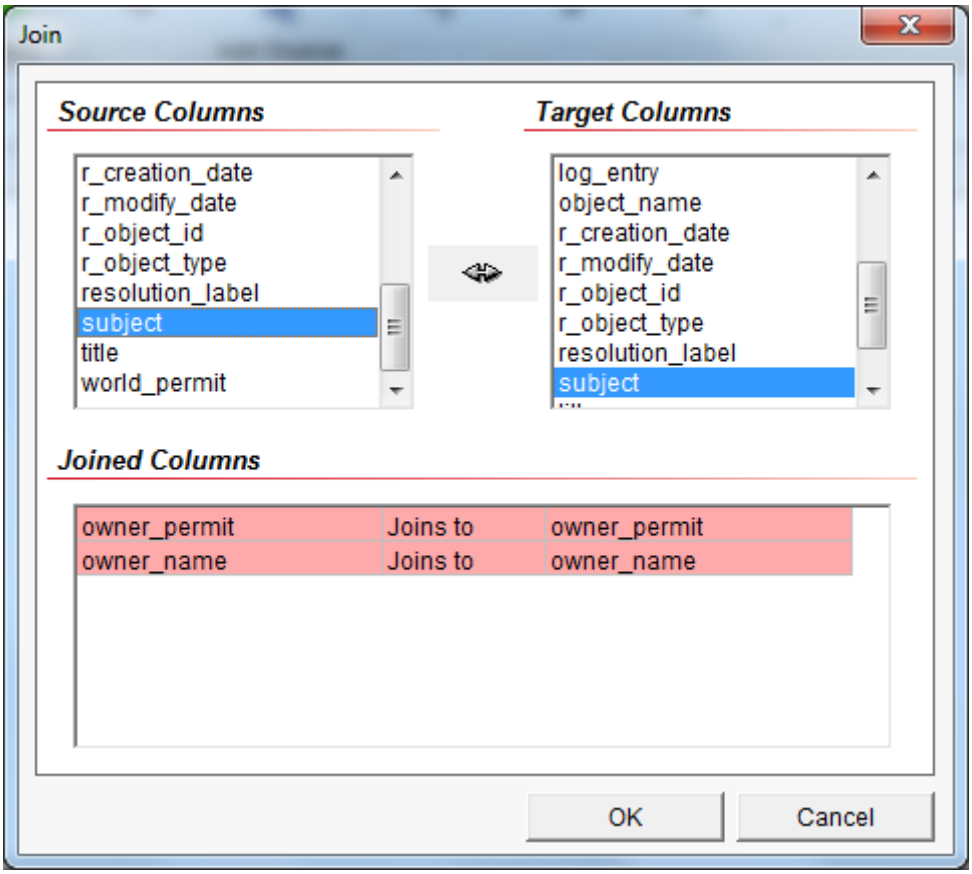


Figure 21 – Joining columns

To join to columns, select the columns in both lists and click on the button *Join*. Columns can also be dragged and dropped on the opposite list. To delete a join, use the context menu of the *Selected Columns* grid.

After successful joining, placeholders can be added to the script template from the context menu or value lists. They look like:

```
{data:object_name}
```

11.6. Generating a Script

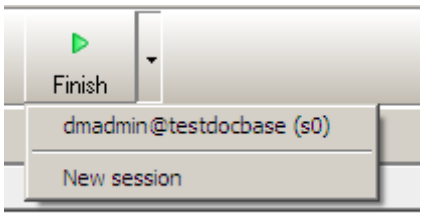
The script can be generated from the template by clicking on the *Generate* button. During generation, the button *Stop* is activated to stop the generation process and return to the script template.

If the script is finished, the label changes to *Template*. A click brings you back to the script template.

A generated script can be saved or executed.

11.7. Executing a Script

The *Execute Script* copies the script into the appropriate script view of the active dqMan session window, or, depending on the script type, opens the Docbasic interpreter.



If the script should be executed in a different session, the available sessions will be listed in a submenu that is opened by clicking on the downward arrow button.

A new session can also be established from this menu.

The script generator will be closed automatically.

11.8. Type dependent Script Templates

Type dependent script templates are a special variant. They are bound to a certain Documentum object type, identified by the type tag. This type of script template will be available directly in the DQL result list context menu, if the list contains a column with `r_object_id` values.

Additionally, these script templates do not depend on values of the result list. Only the `r_object_id` is obligatory. All other attribute values are loaded from Documentum during the generation process.

Type dependent script templates can be generated with the script generator. They must be saved in the subdirectory ".\Scripttemplates" within the dqMan program directory. Furthermore, during save of the script, the type Typescript has to be chosen in the save file dialog. At last, the template has to be named and a type tag must be selected in an additional dialog.

Additional data sources are not allowed in type dependent script templates.

dqMan includes an example, the "ACL Create Script“:

Script Header:

```
# ACL Create Script Template
# Created 06/26/2004
# ------------------------------------------------------------
```

Script Body:

```
# Create ACL {object_name}
create,c,dm_acl
set,c,l,object_name
{object_name}
set,c,l,description
{description}
set,c,l,owner_name
dm_dbo
set,c,l,acl_class
{acl_class}
{LOOP r_accessor_name,_accessor_name,_accessor_permit,_accessor_xpermit_names}
grant,c,l,{_accessor_name},{_accessor_permit},{_accessor_xpermit_names}
{END LOOP}

save,c,l
# ------------------------------------------------------------
```

Script Footer:

```
# END ACL Create Script
```

Please note the use of the computed attributes *_accessor_name, _accessor_permit* and *_accessor_xpermit_names* in the LOOP instruction. Since computed repeating attributes are computed on demand, they usually return the wrong number of contained values.

To ensure the correct processing of the LOOP instruction, the regular attribute *r_accessor_name* of the type *dm_acl* is included in the loop attribute list at the first position, without being referenced within the loop.

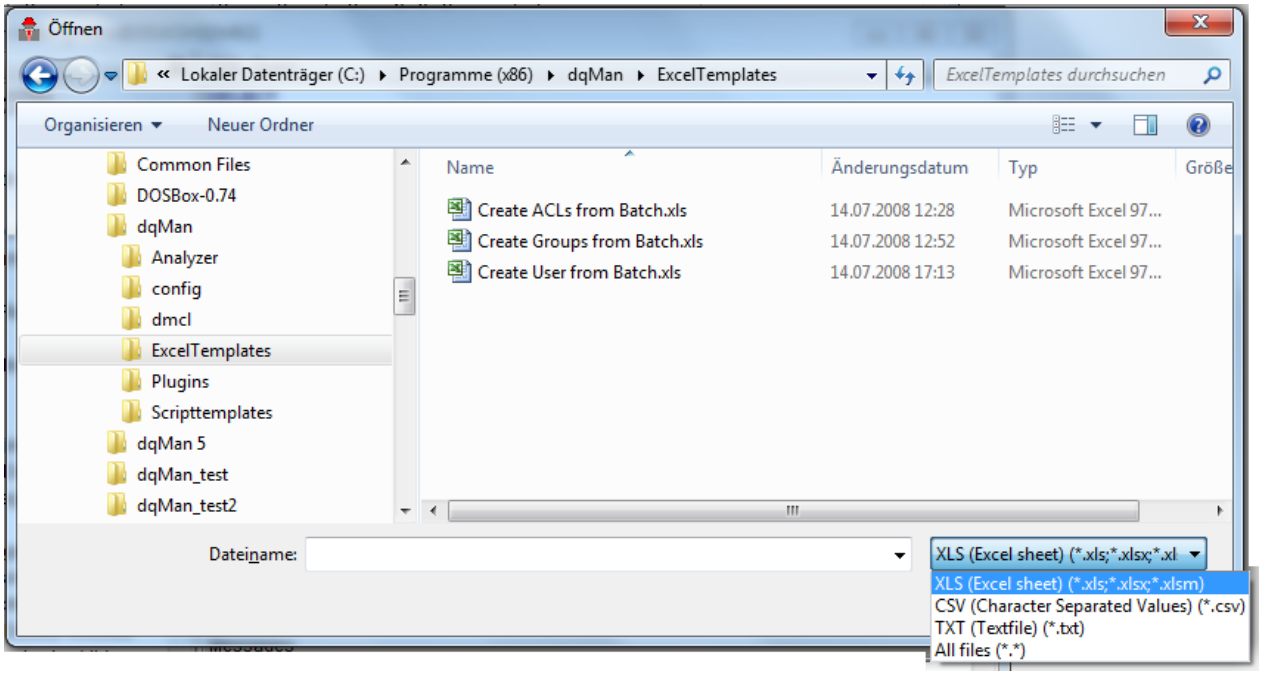
12. Import

The import function allows importing data (no documents) from external sources into a repository.

A data import takes place in two steps. First the data is loaded into a dqMan data grid. In the second step, the data is processed using the script generator.

The Import function is available in the *Extras* menu.

A file open dialog is opened, where Excel files, CSV files or text files can be selected.



Depending on the selected file type, one of the following described import assistants will come up.

12.1. Import of CSV or Text Files

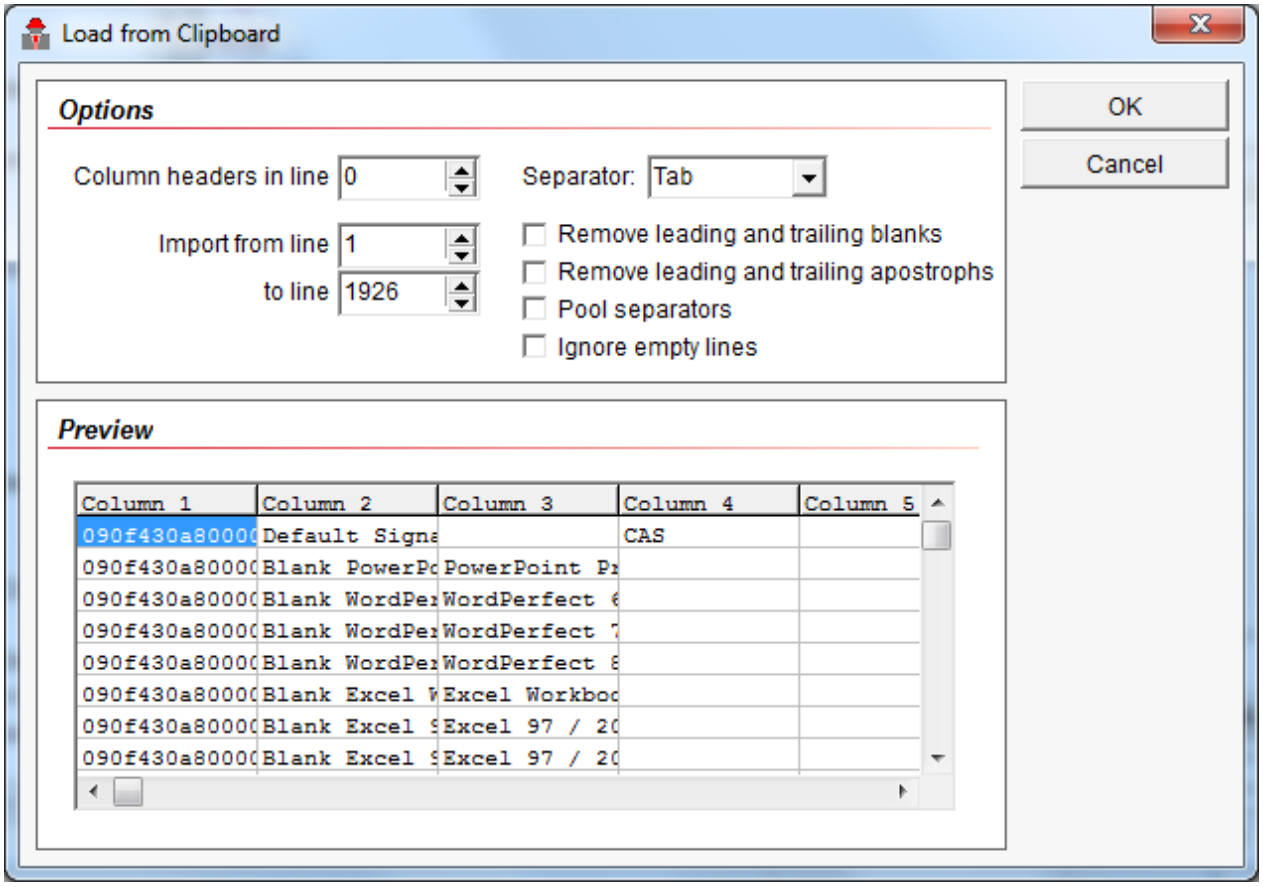


Figure 22 – Data Import Assistant

Header line: Line number with column headers, enter 0 for none.

Start row: Line to import from.

End row: Line to import to.

Separator: The character that separates the data columns. Valid are Tab, Semicolon, Comma or any free combination of characters. Also available is the Fixed Width option.

Options: Remove leading and trailing blanks - removes all surrounding blanks.
Remove leading and trailing apostrophes – removes apostrophes.
Pool separators – combines several sequential separators into one.
Ignore empty lines – lines without data are not imported.

Preview: The preview shows how the selected options affect the import result. The number of lines displayed as preview is limited to 25.

12.2. Import of Excel Files

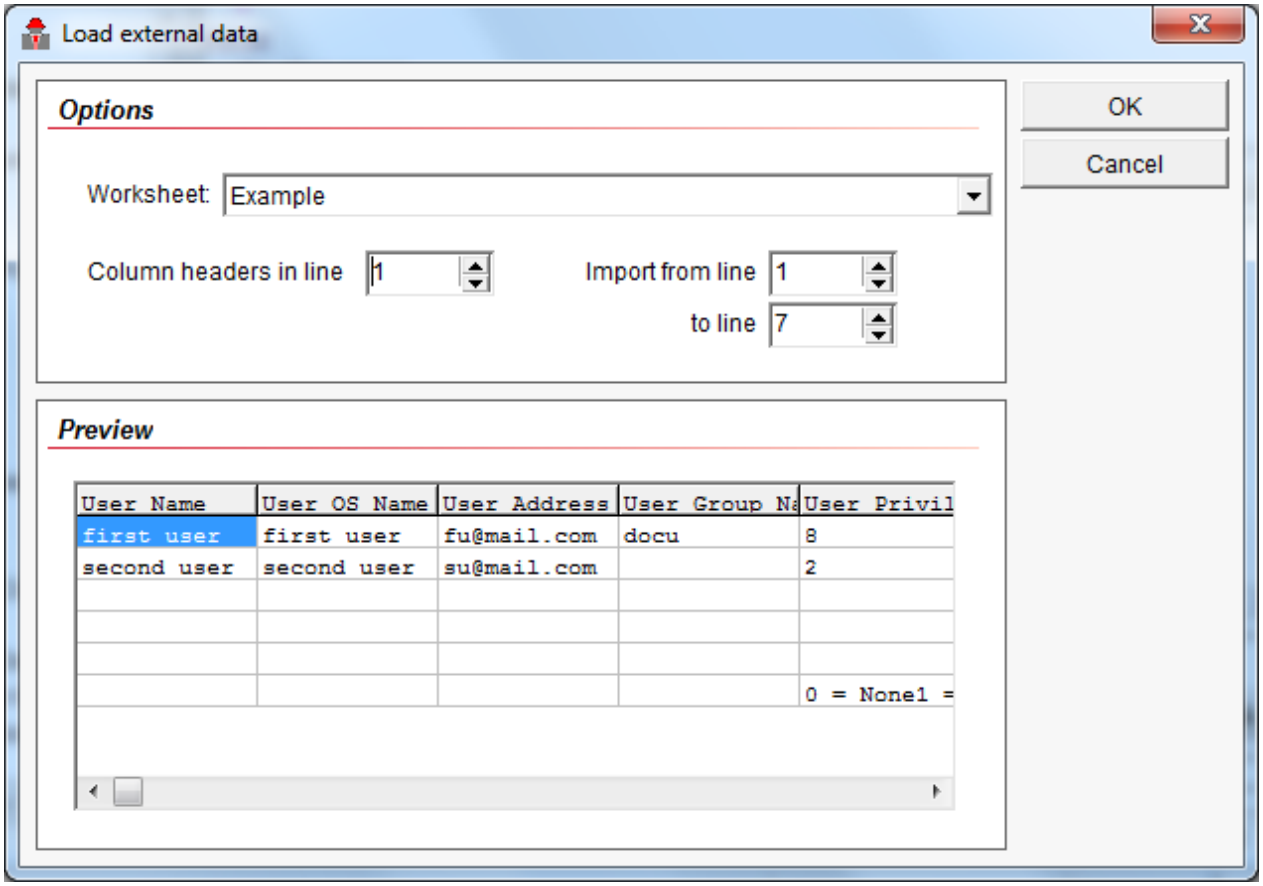

Load external data
Options
Worksheet: Example
Column headers in line 1
Import from line 1
to line 7
OK
Cancel
Preview
User_Name | User_OS_Name | User_Address | User_Group_Na | User_Privil
first user | first user | fu@mail.com | docu | 8
second user | second user | su@mail.com | | 2
0 = None1 =

Figure 23 – Excel Import Assistant

Excel Worksheet:	The worksheet to import from the current opened excel file
Header line:	Line number with column headers, enter 0 for none.
Start row:	Line to import from.
End row:	Line to import to.
Preview:	The preview shows how the selected options affect the import result. The number of lines displayed as preview is limited to 25.

12.3. Import from Clipboard

Data can also be imported directly from the clipboard, if data of type text is available. Calling this function opens the CSV import assistant.

13. Batch Object Creation

The Batch Object Creation Function is an assistant that combines different dqMan technologies for easy creation of objects. The assistant uses a set of predefined templates installed with dqMan. Templates are available for ACL, user and group creation.

The function is called from the *Extras* menu that has a submenu showing all available templates for batch creation.

The first step is to fill out an Excel template with the necessary data. The delivered Excel templates consist of three Worksheets: User Data for the data to be processed, HowTo with a detailed explanation and Example with an example of how to fill out the Excel Template. After filling out the template it should be saved as a different file to keep the template in the original format.

The next step is to import the Excel Sheet into dqMan, using either the assistant or the *Import from File* function from the *Extras* menu.

At last, a script can be created from the imported data. If the assistant is used, the appropriate script template is loaded automatically into the Script Generator. If the Script Generator is called directly, the script template must be loaded manually by clicking on the downward arrow button beside the load button in the toolbar of the Script Generator.

The delivered templates are fully functional examples. New templates can be added easily by creating an Excel template and a matching script template. Both templates must share the same file name, but with different extensions, and must be stored in destined directories, e.g.

Create a template for batch dm_method creation:

The Excel template is stored as
`<dqMan directory>\ExcelTemplates\Create Methods from Batch.xls`

The script template is stored as
`<dqMan directory>\Scripttemplates\Create Methods from Batch.Template`

with the default for `<dqMan directory>` being `c:\program files\dqMan`

If an Excel template exists in the according folder, it is automatically shown by its name in the *Batch Object Creation* submenu; in this case the menu item *Create Methods from Batch* would be added.

14. Compare data

There are two ways to compare data. The *Compare data* function from the *Extras* menu enables you to compare different data lists, e. g. DQL query results with other DQL query results or with imported data lists.

The *Object Compare List* is a list, where single objects of any type can be added from any repository. These objects are compared attribute by attribute and it offers the possibility to edit directly in the list.

14.1. The „Compare Data“ Function

This function is only available if at least two different data lists are opened. These can be results from DQL queries from any repository, or imported data lists. The assistant is described in the following sections:

14.1.1. Settings

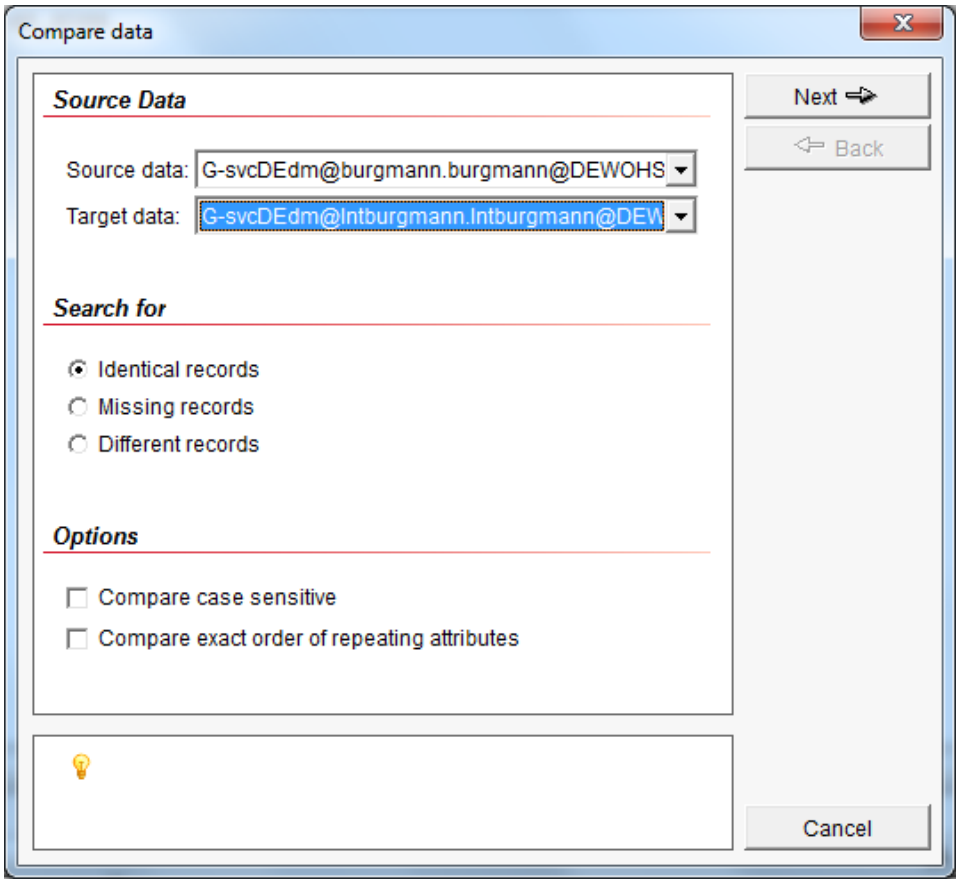


Figure 24 – Comparison Settings

First select the two different data lists to compare. The possible lists are available in dropdown menus. Subsequently, the comparison function is determined. Three functions are available:

Search for identical records: Searches for identical records in both lists.

Search for missing records: Searches for missing records in both lists.

Search for different records: Searches for records, that are joined by one or more column but differ in other attributes.

With the following options the comparison can be influenced:

Compare case sensitive: Compares case sensitive

Compare exact order of repeating attributes: Repeating attributes can have the same values, but in different order.

14.1.2. Data Columns

The two data lists must allow correlation of the data records. At least one column of the source data list must be joined to one column of the target data list.

This is done on the second page of the assistant.

To join to columns, select the columns in both lists and click on the button *Join*. Columns can also be dragged and dropped on the opposite list. To delete a join, use the context menu of the *Selected Columns* grid.

If the function *Search for different records* has been selected, the comparable columns must be defined additionally. This can be done using the button *Compare* or by pressing the key <Ctrl> while dragging and dropping one column on the opposite list.

The search is then done for objects that can be joined by all *Joins to* columns, but that differ in the *Compare to* columns.

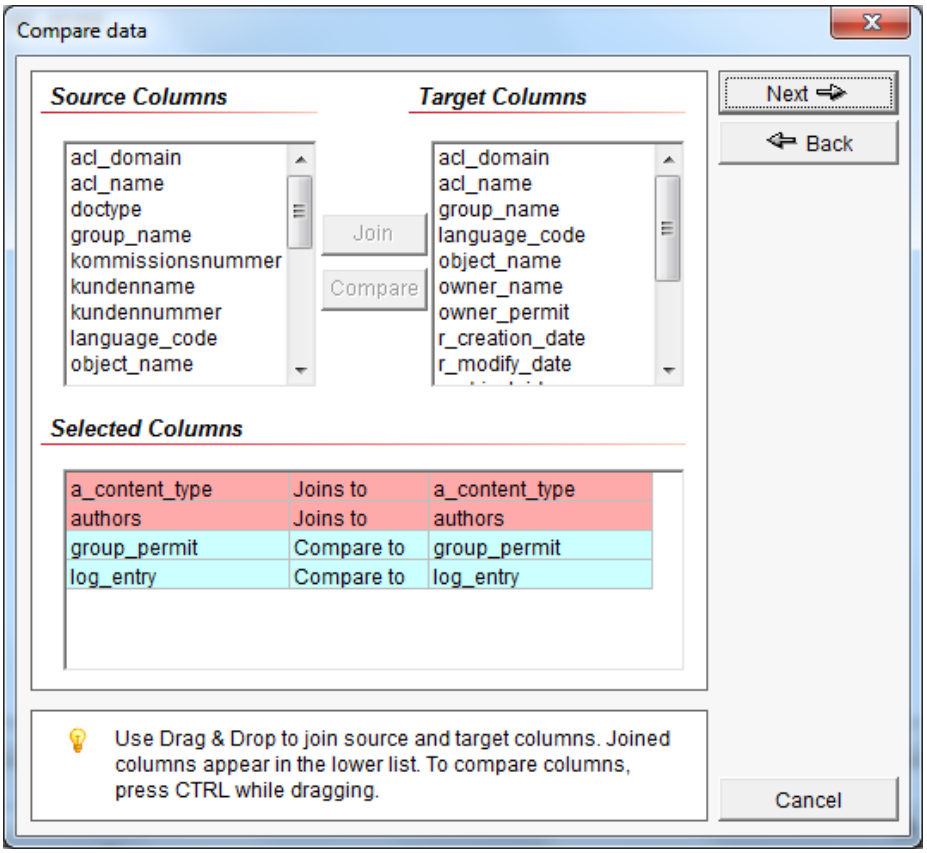


Figure 25 – Joining Columns

14.1.3. Results

On the last page, the result of the comparison will be displayed.

The number of found data records can be different for both lists, since the criteria can be fulfilled by more than one data record

14.1.4. Working with Marked Records

The result of the compare operation will be displayed in the corresponding data lists. All found records will be marked. If the function *Search for different records* was executed, the rows with differences will be highlighted.

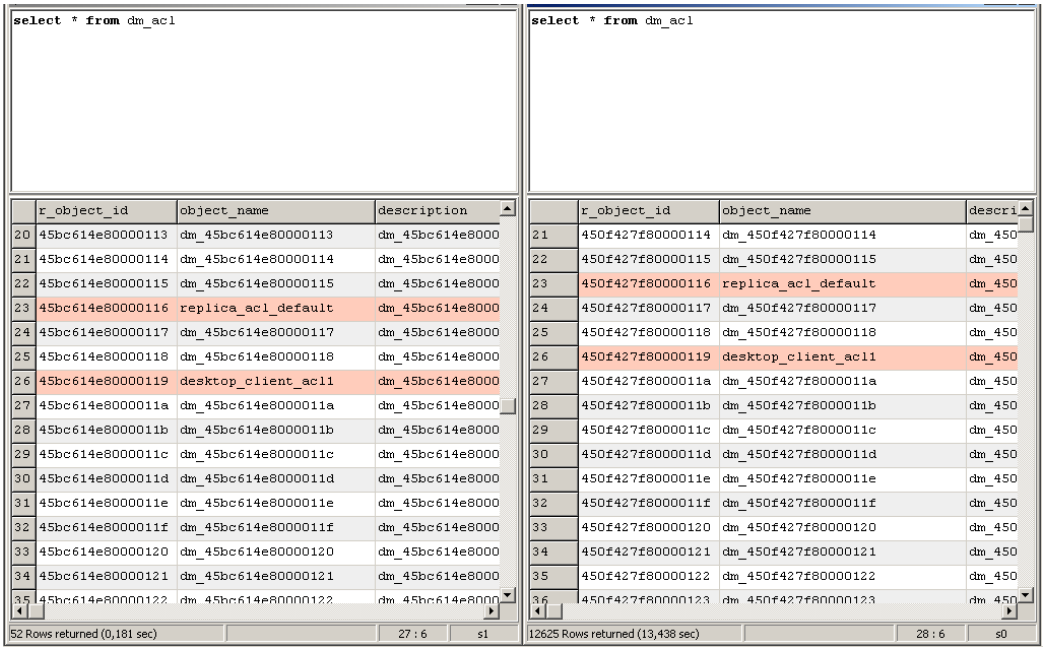


Figure 26 - Working with marked records.

The marks will remain until either a new DQL query is executed in the session window, a new data compare is run or the marks a removed manually, using the function from the context menu.

Marked records can be processed as group, similarly as selected data records. Object functions, export functions and the Script Generator support marked records.

To execute a function for all marked records, exactly ONE cell of any marked record must be selected and the function started. If more than one cell or a not marked cell is selected, the function will be executed for the selected objects/cells.

If the compare was executed with the functions *Search for identical records* or *Search for different records*, the rows in the two grids are linked to each other. A center mouse click on a marked record in one list will scroll the other list to the corresponding record.

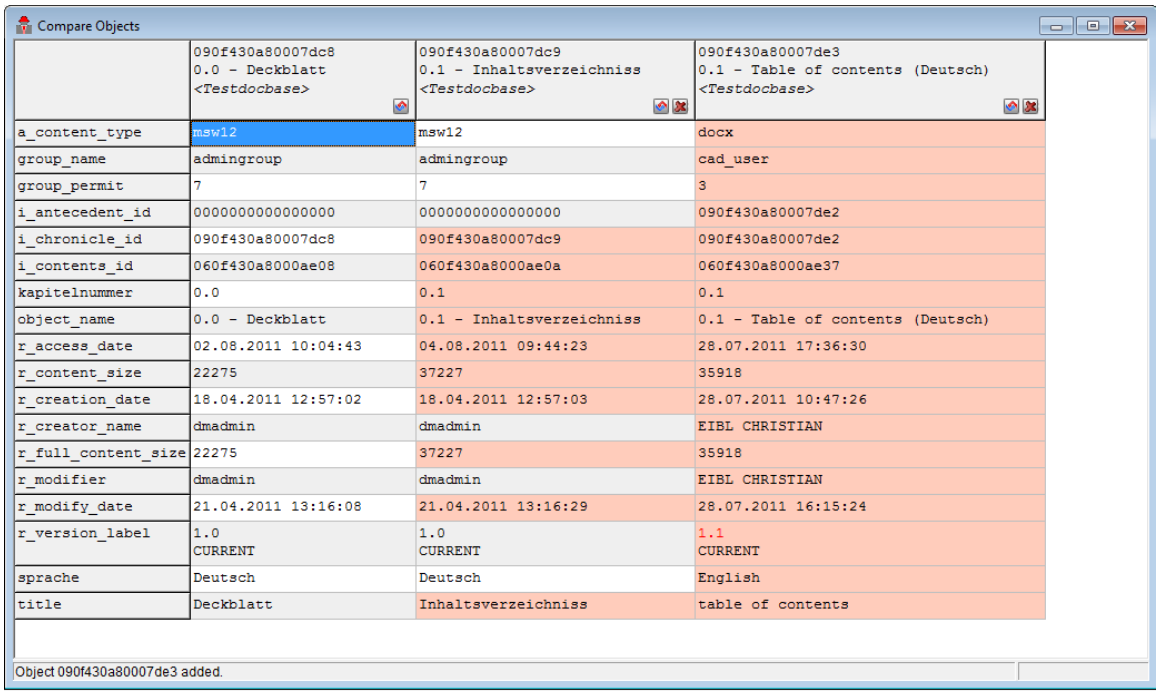
If more than one record from one list is linked to a single record in the other list, repeated mouse clicks on the single record will loop through all linked records successively.

If a *Search for different records* was executed, the function *Add to Compare List* will add the selected object and all the corresponding objects to the Object Compare List in one go if only one object is selected. The Object Compare List will be cleared first.

14.2. The Object Compare List

The *Object Compare List* is a list, where single objects of any type can be added from any repository, e.g. a dm_acl object from repository A could be compared with a dm_cabinet object from repository B.

Objects can be added from the DQL query result list calling the function *Add to Compare List* from the context menu. There can be only one Object Compare list. All objects will be added to the same list. It will be automatically created when the first object is added. Closing the list removes all objects.



	090f430a80007dc8 0.0 – Deckblatt <Testdocbase>	090f430a80007dc9 0.1 – Inhaltsverzeichniss <Testdocbase>	090f430a80007de3 0.1 – Table of contents (Deutsch) <Testdocbase>
a_content_type	msw12	msw12	docx
group_name	admingroup	admingroup	cad_user
group_permit	7	7	3
i_antecedent_id	0000000000000000	0000000000000000	090f430a80007de2
i_chronicle_id	090f430a80007dc8	090f430a80007dc9	090f430a80007de2
i_contents_id	060f430a8000ae08	060f430a8000ae0a	060f430a8000ae37
kapitelnummer	0.0	0.1	0.1
object_name	0.0 – Deckblatt	0.1 – Inhaltsverzeichniss	0.1 – Table of contents (Deutsch)
r_access_date	02.08.2011 10:04:43	04.08.2011 09:44:23	28.07.2011 17:36:30
r_content_size	22275	37227	35918
r_creation_date	18.04.2011 12:57:02	18.04.2011 12:57:03	28.07.2011 10:47:26
r_creator_name	dmadmin	dmadmin	EIBL CHRISTIAN
r_full_content_size	22275	37227	35918
r_modifier	dmadmin	dmadmin	EIBL CHRISTIAN
r_modify_date	21.04.2011 13:16:08	21.04.2011 13:16:29	28.07.2011 16:15:24
r_version_label	1.0 CURRENT	1.0 CURRENT	1.1 CURRENT
sprache	Deutsch	Deutsch	English
title	Deckblatt	Inhaltsverzeichniss	table of contents

Figure 27 - The Object Compare List

The attributes are aligned vertically. The object in the first column is the master object. All other objects are compared with the master object. Differences will be highlighted. Differences in single values of repeating attributes will be displayed with red font color.

To change the master object, the new master object can be dragged to the first column using the mouse.

If different object types are compared, attributes that are invalid for the object type remain grey and empty.

The button refreshes the values of a column, deletes the column.

The context menu of the list offers additional functionalities:

Edit:	Values can be directly edited within the table. A change is stored only if it is confirmed with the key <Enter>. Repeating attributes can be edited within a small sub grid. The sub grid offers an own context menu for inserting, deleting or reordering values of the repeating attribute.	
Copy:	Copies all selected cells into the clipboard.	
Dump Object:	If the selected cell contains a valid object id, this id can be dumped.	
Goto …:	Searches for attributes in the list.	
Show Attributes:	All:	All attributes of all object types in the list.
	Master:	All attributes of the master object.
	Type specific:	Type specific attributes of the master object type only.
	Identical values:	All attributes with identical values in every column.
	Different values:	All attributes with a different value in any column.
Compare settings:	Compare case sensitive:	Compares case sensitive
	Exact order of repeating attributes:	Repeating attributes can have the same values, but in different order.
Order Attributes by:	Name:	Sort attributes by name
	Category:	Sort attributes by category, like an object dump
	Not sorted:	Attributes are shown in content server generated order.

.

15. Script Assistants

Script assistants support the development of complex API or DQL scripts.

15.1. Dump & Load

The dump & Load Script assistant can be opened from the *Extras* menu. The available script types can be selected from a submenu. The result script will be pasted into the API Script View of the active session window.
.

15.1.1. Dump entire docbase

Step 1:
In the first step of the assistant the full path to the dump file must be entered. Since this path references the local directories on a remote server, it must be entered manually.

Step 2:
On the second page, the options can be determined. Please refer to the Documentum *Content Server Administrator's Guide* for more information about Dump options.

Step 3:
The last page displays the generated script. It can be edited if necessary. Click the button Finish to paste the script into the API script view of the active session window for execution.

15.1.2. Dump Specific Objects

In order to dump only specific objects, the function *Dump specified objects* can be used.

Step 1:
In the first step of the assistant the full path to the dump file must be entered. Since this path references the local directories on a remote server, it must be entered manually.

Step 2:
On the second page, the desired object types can be selected. To specify certain objects of the selected type, a *Where Clause* can be added. The keyword *Where* is generated automatically, if a clause is not empty.

Step 3:
On the third page, the options can be determined. Please refer to the Documentum *Content Server Administrator's Guide* for more information about Dump options.

Step 4:
The last page displays the generated script. It can be edited if necessary. Click the button Finish to paste the script into the API script view of the active session window for execution.

15.1.3. Load Docbase

Step 1:
In the first step of the assistant the full path to an existing dump file must be entered. Since this path references the local directories on a remote server, it must be entered manually.

Step 2:
On the second page, the options can be determined. Please refer to the Documentum *Content Server Administrator's Guide* for more information about Load options.

Step 3:
The third page displays the generated script. It can be edited if necessary. Click the button Finish to paste the script into the API script view of the active session window for execution.

15.2. Copy Registered Table

The *Copy Registered Table* assistant permits the complete copying of registered tables. The result script can create a database table, register it and copy the content. Each part can be executed independently from one another. It is therefore also possible to copy data between two existing tables only.

This assistant can be found in the *Extras* menu, but it is only available if called from a session window with an active session. It is also available in the Repository Navigator's Types/Tables tab, if a registered table is selected.

The generated script can be saved or pasted into the DQL view of an existing or new session window for execution.

Selection of registered tables and operations

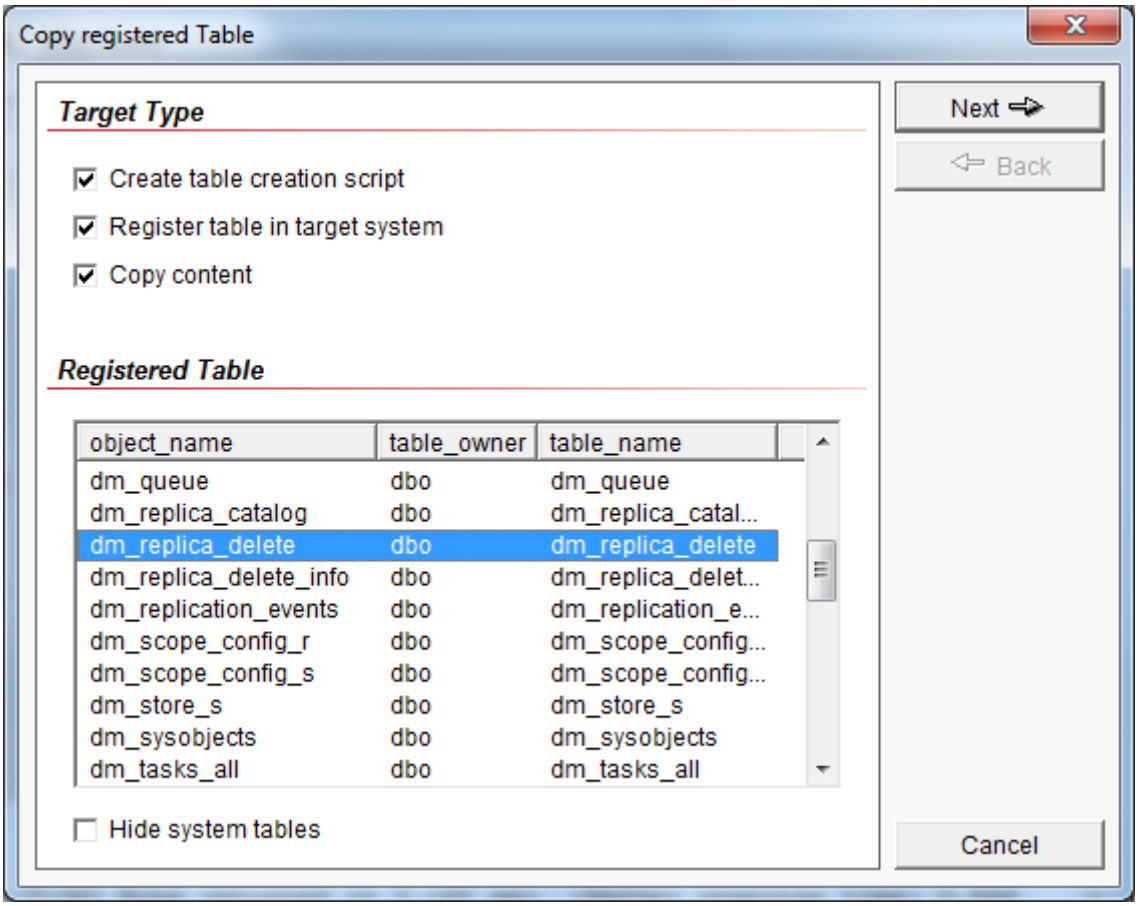


Figure 28 – Copy Registered Table Assistant (1)

On the first page, the registered table and at least one operation must be selected.

Available operations are:

Create table creation script:	Creates the table in the database of the target repository. The table must not exist in the target database.
Register table in target system:	Creates *Register Table* commands that will copy the Documentum registration of the source table. The table must exist in the target database.
Copy content:	Generates a script to copy the data from the source to the target table. The table must exist in the target database and it must be registered in the target repository.

The *Hide system tables* checkbox removes all system tables from the list.

Selecting the target database type and columns

This page is available, if the operation *„Create table creation script"* has been selected.

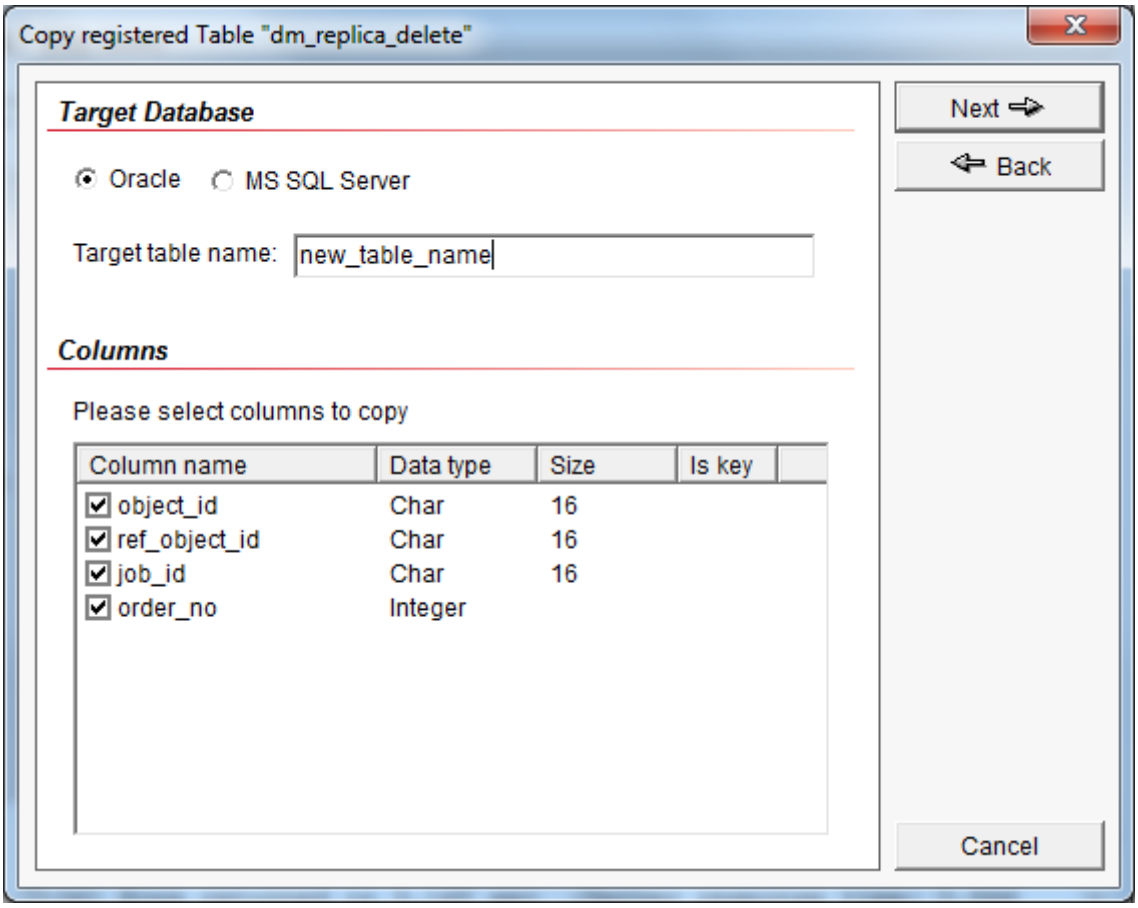


Figure 29 – Copy Registered Table Assistant (2)

The target database type must be selected, in order to generate the correct SQL commands for table creation.

On this page the created table can be assigned a new name by filling in a valid table name into the *Target table name* field.

Further, single columns from the source table can be excluded. If columns are deselected, they will not be copied, registered or filled with data. At least one column must be selected.

Registration and permissions

This page is available, if the operation *„Register table in target system"* was selected.

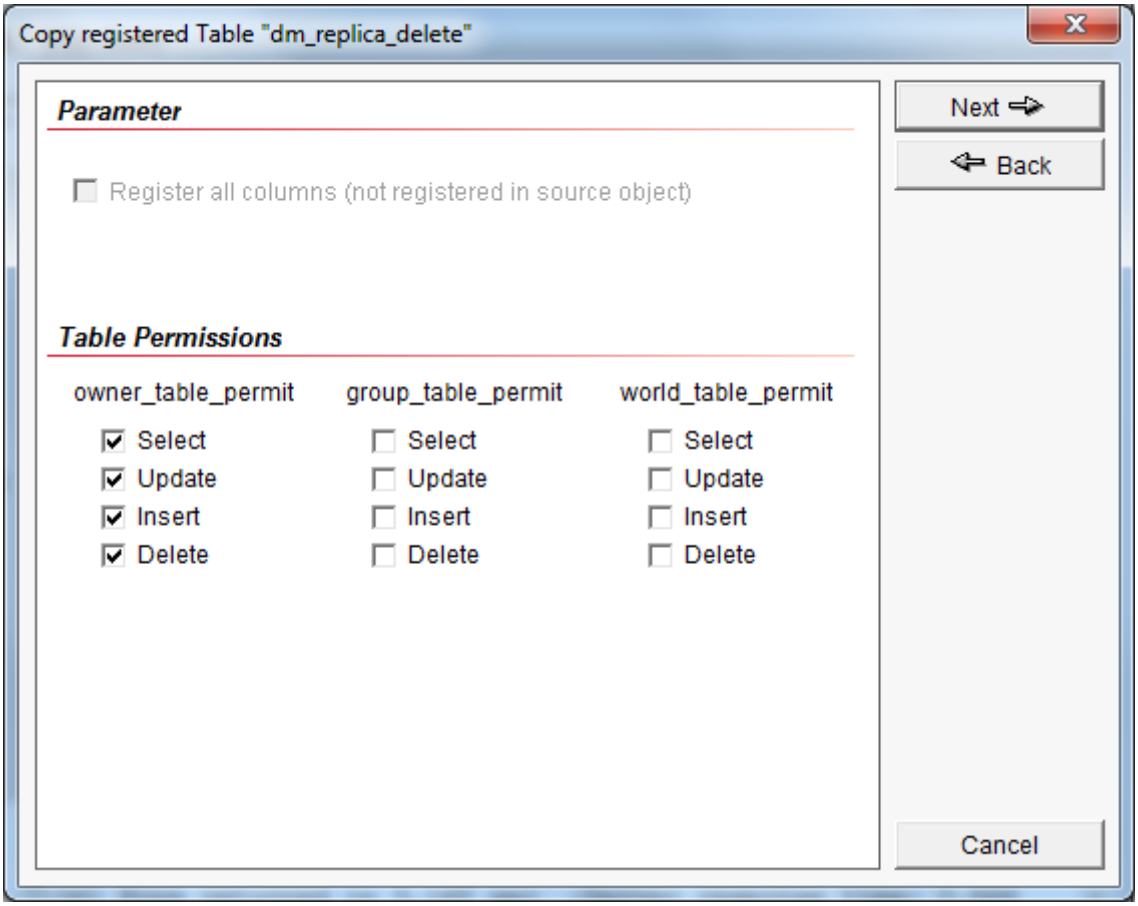


Figure 30 – Copy Registered Table Assistant (3)

The option *Register all columns* will do a complete registration of all copied columns with names and types. This option is only enabled for selection, if the table is not fully registered in the source repository. If it is checked, the registration will differ between source and target repositories.

The table permissions are initialized with the permissions of the source table, but can be changed if necessary.

Data selection

If the operation *„Copy content”* was selected a where clause can be added for data selection.

Result

On the last page, the generated script will be displayed. It can be saved by clicking on the button *Save script* or it can be copied into the DQL script view of an existing or a new session window for execution, using the button *Execute in... .*

The assistant will close automatically if the script is copied into a session window.

15.3. Change Objecttype

Using the Change Objecttype assistant, the object type of objects can be changed. It can be found in the *Extras* menu, but it is only available if called from a session window with an active session and the DQL command view selected. Furthermore, a DQL query result must be available that includes the `r_object_id` attribute.

All selected objects in the result list must be of the same object type. Supported types are dm_document, dm_folder and dm_cabinet with their descendants.

The generated script can be saved or pasted into the DQL view of the current session window for execution.

The first step is to select a new type for the objects from the available types in the current repository. When a new type is selected from the type tree, the necessary migration path is displayed at the bottom of the window. The *Next* button will be activated.

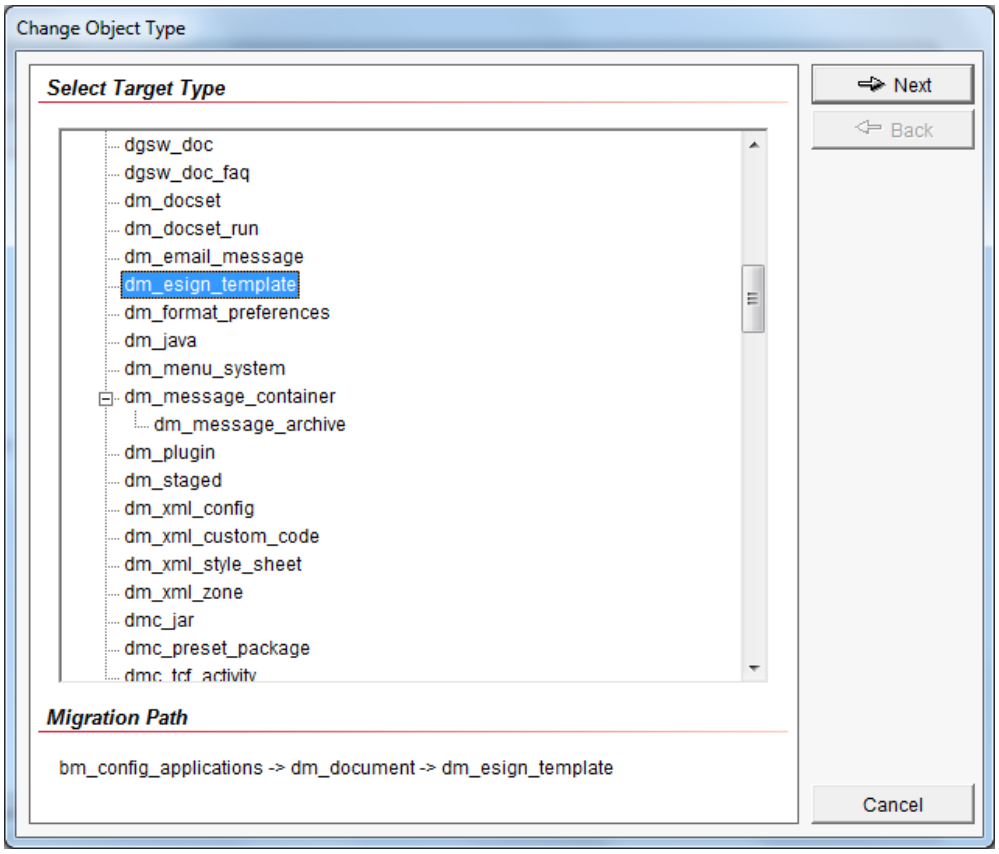


Figure 31 – Change Objecttype Assistant (1)

The next step is to map all attributes from the old type that will be lost through the change in type, to attributes of the new type. These attributes are displayed in the left column of the attribute mapping window.

Selecting an attribute of the new type from the dropdown menus on the right will create the mapping between the old and the new attribute. All attributes including their values that are not mapped will be lost.

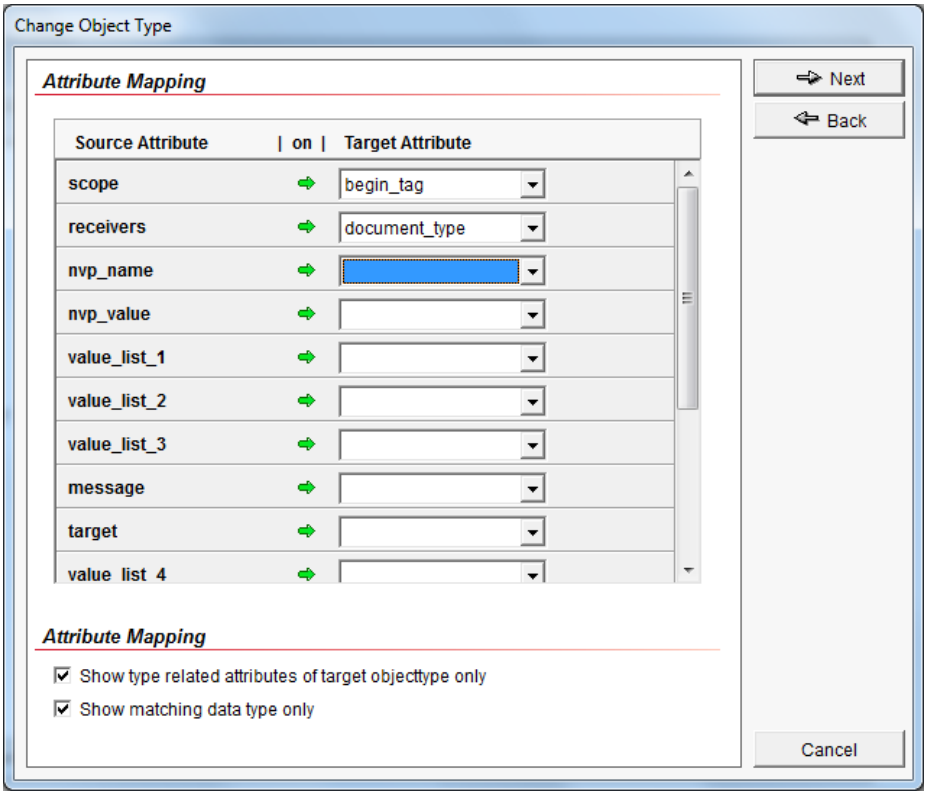


Figure 32 – Change Objecttype Assistant (2)

In the dropdown lists only attributes of the new type will be displayed, that are type specific of the new type and that have the same data type as the old attribute. Additionally, repeating attributes of the old type can be mapped to repeating attributes of the new type only. Single attributes can be mapped to single attributes only.

Changing the options can influence this behaviour:

Show type related attributes of target object types only:

Deactivate this option to select from all attributes of the new type.

Show matching data type only:

Deactivate this option to map different data types.

On the last page, the generated script will be displayed. It can be saved by clicking on the button *Save script* or it can be copied into the DQL script view of the current session window for execution, using the button *Finish*.

On the third page some optional features can be activated:

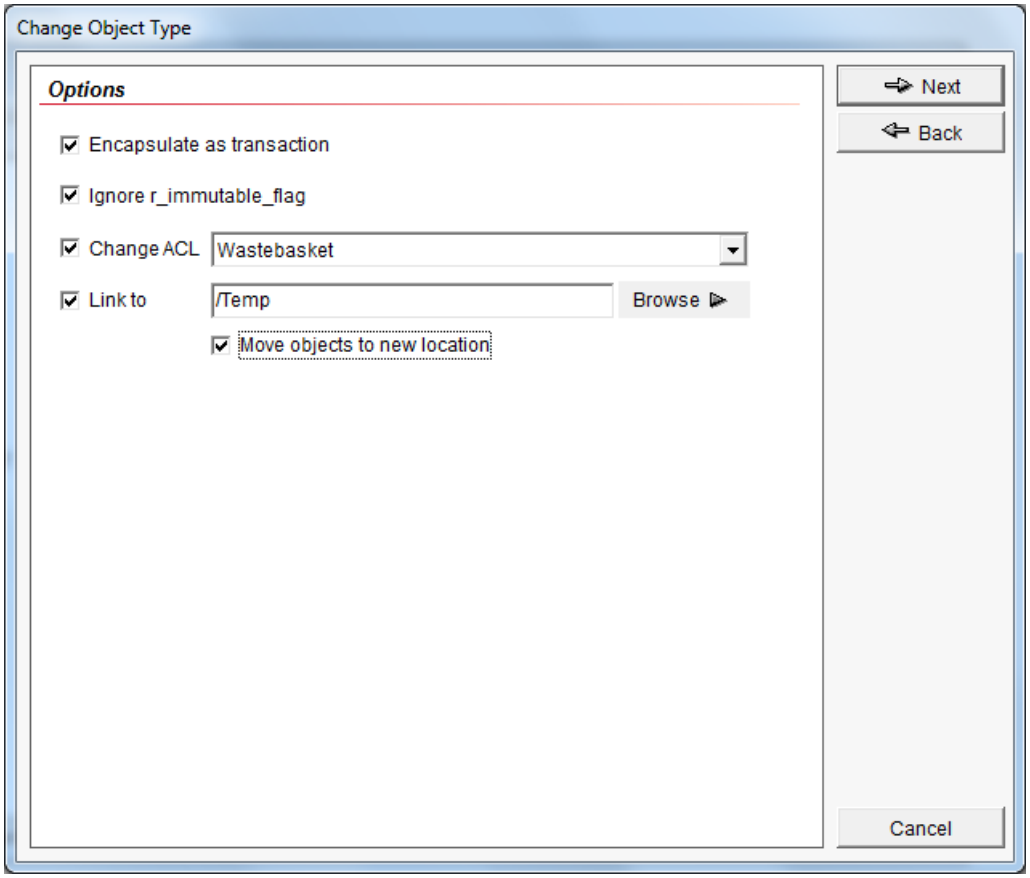


Figure 33 – Change Objecttype Assistant (3)

Encapsulate as transaction:	The complete change of the object type for each object is treated as one transaction. If an error occurs, the action will be rolled back.
Ignore r_immutable_flag:	Older versions are usually protected from being changed by the r_immutable_flag. This behaviour can be ignored.
Change ACL:	The changed object can be assigned a new ACL in one step.
Link to:	Changed Objects can be linked or moved to a new folder in one step.

16. Options

The *Options* are the central tool in order to adapt the features and functionalities of dqMan. They are available from the *Extras* menu. The window shows a tree structure on the left side to navigate the different categories.

16.1. General

The General category has four subsections:

16.1.1. Settings

- Open session at startup: Automatically opens a session window at start up. The start view can be selected from the *Start with* dropdown.
- Restore last executed query from history: Restores the last executed DQL query from the history.
- Show toolbar: Displays the toolbar.
- Show tabs for open windows: Displays an additional toolbar with buttons for all open windows.
- Save History Data: Saves DQL and API history data on local disk for later use. *History size* limits the size to the selected value. This limit is valid per session window.
- Autosize row heights: Auto sizes the cells in the history list to *Max Rows* lines, if multi line commands are displayed.

16.1.2. Login

- Save login info: Saves the entered login information, either with or without password.

  If the option *always show next unused login* is activated, the login dialog will be initialized with the last used login data from the login history, depending on the available repositories.
- Dump object with Autologin: If a dump from a repository without valid session is requested, a login from the login history will be used to acquire a valid session.

- Use full qualified repository names: If checked, full qualified repository names (`repository.content_server_name@host_name`) are used for repository identification.
- Order login history by: Sorts the displayed order of saved logins in the login history of the login dialog.
- Edit history: Edit the saved logins in the login history.

16.1.3. Shared Favorites

Object Functions and the DQL favorites are stored locally. Therefore each instance of dqMan uses its own data.

But they can also be shared if the full paths to the shared favorite files are supplied. The XML files which contain the favorites and Object Functions (DQL-Repository.xml and Functions.xml), must be copied to a network drive to be shared.

If the shared files are not available, the local versions will be used instead.

16.1.4. Documentation

To fill the *Documentation* menu of dqMan, links to documents, URLs or program files can be defined in this category.

The function bar offers functions for managing the entries:

Add a document. If a URL or program file should be entered, it must be typed into the cell manually.

Delete the selected entry.

Add a menu separator

Move selected entry upwards

Move selected entry downwards

16.2. API Command View

- Show API hints: Shows hints with syntax information for API commands.
- Clear input fields: After successful execution of a command, the input controls will be emptied. In case of an error, the command remains for correction.
- Reversed API log direction: Adds new lines at the top of the log.

16.3. DQL Command View

16.3.1. Settings

- DQL syntax help: Enables syntax help for DQL commands and functions.
- Auto indent: Indents the cursor in new lines. Leading tabs and spaces will be copied from the proceeding line.
- Restore column order and width: If the names and number of columns are not changed, the DQL result list settings (column width, column order and visible columns) will not change if DQL queries are executed again.
- Reverse displayed order of repeating attributes:

  DQL queries usually return values of repeating attributes in reversed order. This option reverses the order again.
- Move focus to DQL Result List after query execution:

  Moves the focus to the DQL Result grid, if a query was executed successfully. If not checked, the focus remains in the DQL area.
- Autosize columns: Column widths are adapted to the length of the represented data, with the limitation defined in the *Max. Characters* field.
- Autosize rows: Row heights are adapted to the max. number of lines of repeating attributes, with the limitation defined in the *Max. Rows* field.

  This option does only apply, if the DQL query contains repeating attributes and their values are separated by a carriage return.
- Separate repeating attributes: Values of repeating attributes can be separated by a comma, a semicolon, a carriage return, a tab or by a custom string.
- Enclose single values with: Encloses values of repeating attributes either with single quotes or with double quotes or with no character at all if nothing is selected.

16.3.2. Colors

- Highlight even lines (background): Background color for even lines.
- Marked lines (background): Background color for marked rows.
- Different columns (background): Background color for cells with different content (from Compare data function)

16.3.3. Syntax Highlighting

On this page the range, color and style of the syntax highlighting can be customized. Depending upon the selected level more or less reserved words are marked.

Available levels are None, Low, Medium and All. An Example can be seen in the last column.

Syntax highlighting is available in the DQL View only.

16.3.4. Grid Hints

Grid Hints are hints that show additional information for single objects. They are faded in if the mouse is moved onto the row header (containing the row number).

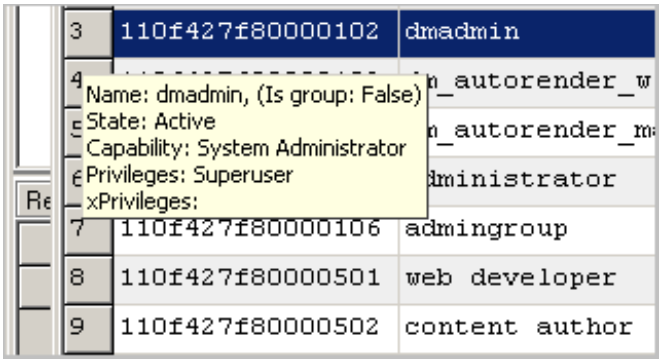


Figure 34 – Grid Hint for a dm_user object

Grid Hints can be defined for different object types. They consist of a hint template and translations.

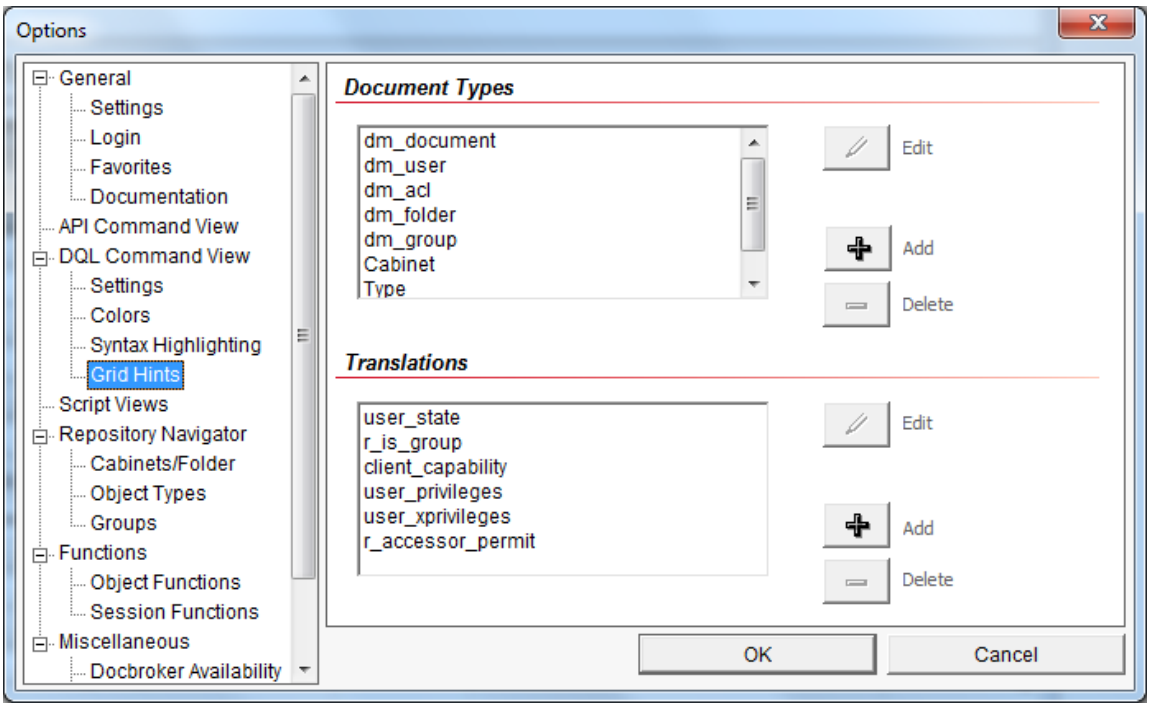


Figure 35 – Grid Hints

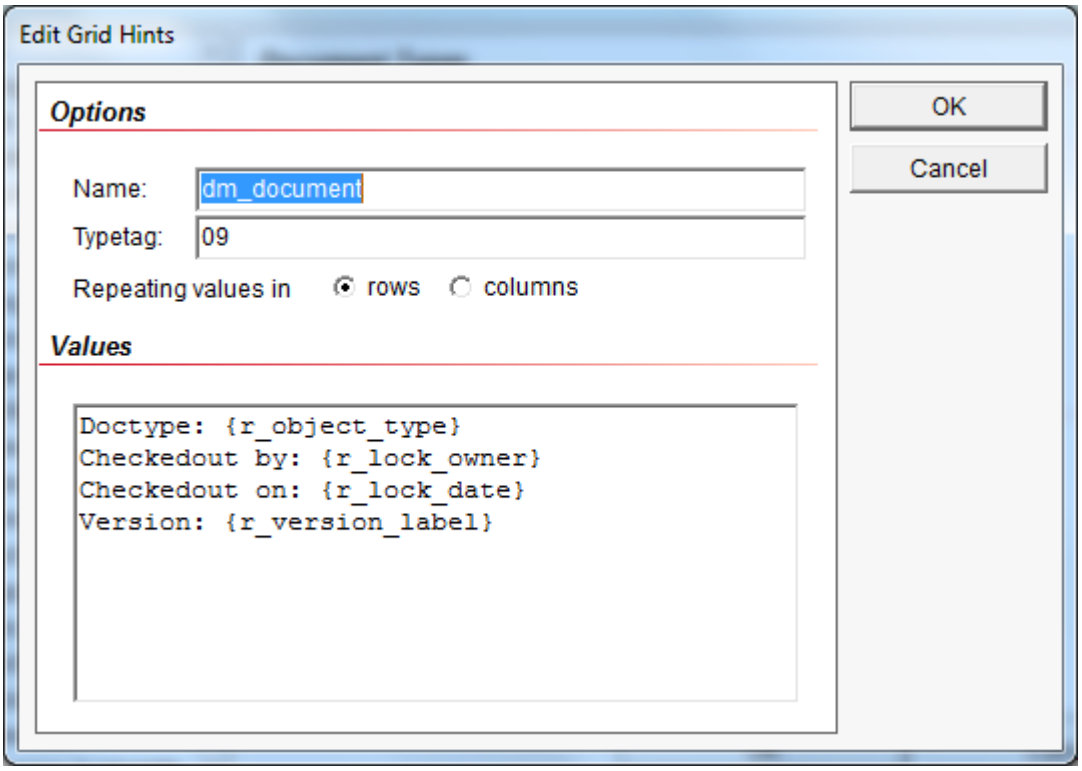


Figure 36 – Editing the Grid Hints

A Grid Hint template contains the following elements:

- Name: Freely selectable name, shown in the grid hint list.
- Typetag: Single Documentum type tag (e.g. 09 = dm_document). Instead of a type tag a type name can be entered that will be replaced with the type tag automatically.
- Values: Hint template. References to attributes must be enclosed in curved clips and will be replaced automatically. All valid attributes of the type can be referenced. All other text and invalid attributes will be copied without changes.
- Display repeating values in: Repeating values can be displayed in columns (whole template line is copied for each value in the attribute) or rows (values will be separated by a comma).

  If more than one repeating attribute is in the same line and displayed in *column mode*, the individual value indices are correlated (see Grid Hint for dm_acl objects).

Translations can translate values into readable designations. They are valid global, i.e. a translation for the attribute `client_capability` is valid for all defined Grid Hints.

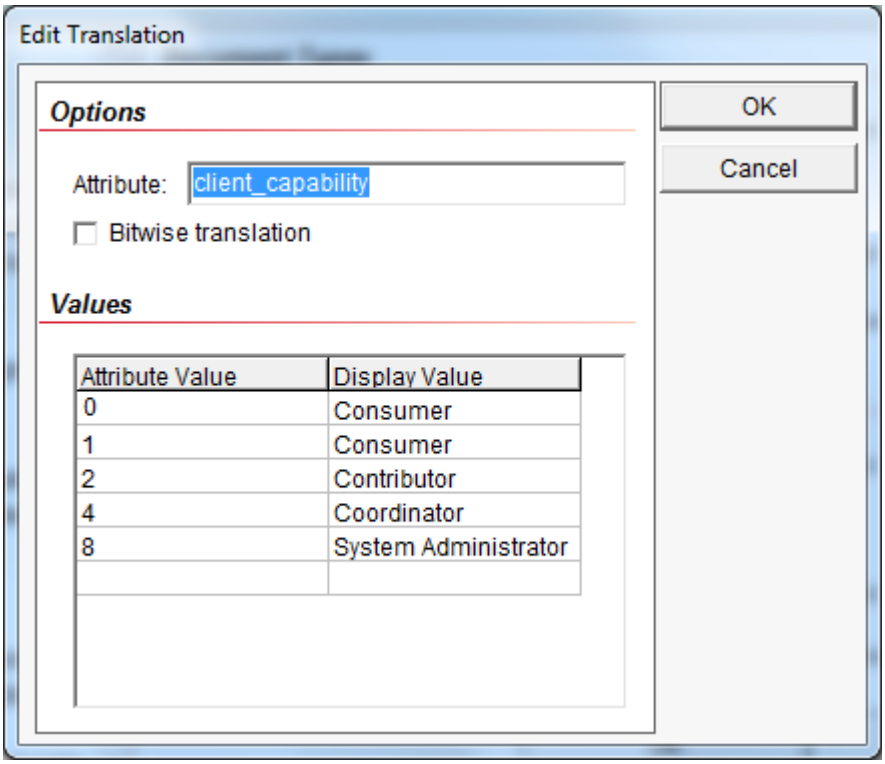


Figure 37 – Grid Hints Translations

- Attribute: Name of the Documentum attribute.
- Bitwise translation: Bitwise coded values, e.g. the attribute user_privileges of the type dm_user, can be translated using this option. Multiple values will be separated by a comma.
- Values: Mapping of attribute values to displayed designations.

16.3.5. Repository Navigator

16.3.5.1. Cabinets/Folder Tree

Auto expand at startup: The cabinet level will be loaded and displayed on login.

Hide private cabinets: Private cabinets of other users will be hidden. However, private cabinets of the current user are always visible.

Autoformat queries: Generated queries from query templates will be formatted automatically.

Clear histories: Clicking this button clears the histories for the functions *Create Cabinet/Folder* and *Create Document*.

Query templates: Templates to generate the DQL queries. They appear in the context menu of the repository tree.

A query template consists of:

Name: Freely selectable name, shown in the context menu.

Query: Template of the DQL query. Where ever the code {*} occurs, it will be replaced with the current folder path of the selected folder in the repository tree.

```
Select * from dm_document where folder ('{*}')
```

Additional options are available:

Use as default

One standard template can be defined by enabling this option. The standard template will not appear in the context menu but is called directly with the left or center mouse button.

Enable direct execution:

If this option is enabled the generated DQL query will be executed immediately. This sort of template is marked with a green arrow in the context menu.

16.3.5.2. Object Type Tree

Auto expand at startup:	The first level will be loaded and displayed on login.
Autoformat queries:	Generated queries from query templates will be formatted automatically.
Show customized types only:	Only customized types and all types with customized subtypes will be shown in the tree. Also all Documentum base type tables will be hidden, e.g. dm_sysobject_s, whether they are registered or not.
Start with:	Object type to start with. (Type tree only)
Query templates:	There are two kinds of query templates, for dm_type objects and for registered tables. The functionality has already been described in the section → *Cabinet/Folder Tree*

16.3.5.3. Group Tree

Auto expand at startup:	The first level will be loaded and displayed on login.
Autoformat queries:	Generated queries from query templates will be formatted automatically.
Group top level by first <xx> characters:	A repository can contain a large number of groups. This option lets you create an additional first level in the group tree for a better overview, by grouping the group names by their first <xx> characters.
Limit no. of groups to:	This option limits the number of queried groups.
Query templates:	*(→ see Cabinet/Folder Tree)*

16.4. Script Views

Clear script log at restart:	The script log area is cleared each time a script is executed.
Copy comments to script log:	Comments in the script will be copied to the script log.
Add line numbers to script log:	The line number of the command will be added to the script log. If line numbers are stored, the function *Show command* from the context menu allows jumping from any line in the log to the corresponding command in the script.
Synchronize log to file:	This options allows copying all log messages into a file. If this option is selected, the user has to select a target file every time a script is started. If an existing file is selected, the additional content will be appended to the existing file.
Log everything:	Everything will be logged.
Log results only:	Only results or error messages will be logged.
Log errors only:	Only error messages will be logged.
Log nothing:	Nothing will be logged.
CSV format for query results:	(*Only valid for DQL script log)* Results of DQL queries will be logged in CSV format.
Fixed width for query results:	(*Only valid for DQL script log)* Results of DQL queries will be logged as list with fixed width. Values can be truncated, if too long.
Next line:	Color of the next script command to be executed (in debug mode only).
Error:	Color of an incorrect script command (in debug mode only).

16.5. Functions

Functions are user programmable extensions for dqMan. There are two different types of functions: Object Functions and Session Functions.

Object functions are available from the DQL result grid. They can contain references to the DQL result grid, as known from the Script Generator. On execution the function will be executed once for every selected object whereby the references are replaced with values from the current result grid.

Session functions need a valid session and are executed only once. They cannot reference any values from the current session window.

dqMan is installed with a number of object and session functions that demonstrate the use of the extended functionality described below. If there are any more questions, please send an email to dqMan@fme.de.

16.5.1. Object Functions

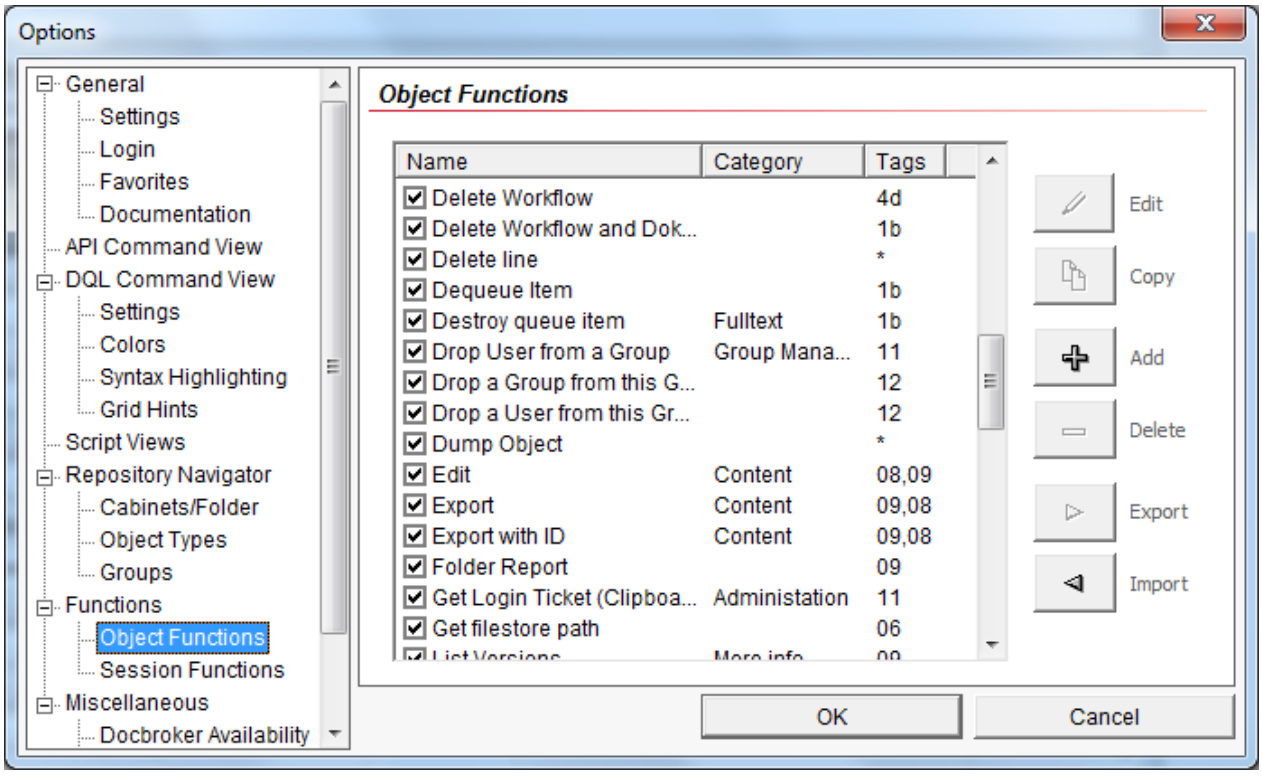


Figure 38 – Object Functions

This page lists all available functions and offers tools to manage them.

Edit a function

Copy a function

Add a new function

Delete a function

Export selected functions to XML for exchange with other dqMan users.

Import functions from XML file.

Object functions can be deactivated without deleting them. In this case the function name is unchecked.

The Object Function Assistant

For easy creation and editing an assistant is available.

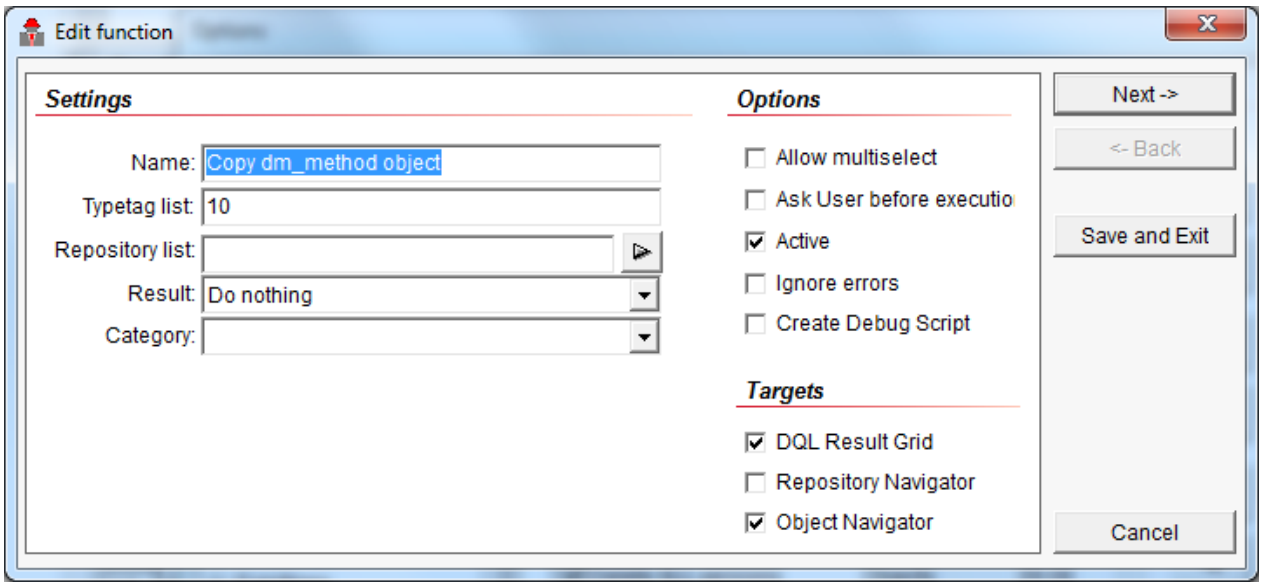


Figure 39 – User Function Assistant (1)

- Name: Name of function, shown in the context menu.
- Typetag list: Comma separated list of Documentum type tags (e.g. 09 = dm_document) to restrict the function to certain object types. Instead of type tags, type names can be entered that will be replaced by type tags automatically.

  To assign a function to all object types, the character * can be entered.

  If no value is entered, the function is always available. But it will be executed only once independently of the number of selected data records. (The example function *Last SQL statement* uses this feature).
- Repository List: Enables the allocation of functions to one or more repositories. Repositories can either be added manually, or selected from a repository list by clicking on

  Multiple repository names must be separated with a comma.

  An exclamation mark placed in front of the repository name, e.g. „! testdocbase “, hides this function if this repository is active.
- Result: The result of a function execution can be presented in different ways:

Count processed objects	Shows the number of processed objects in a dialog box.
Show result in dialog	Shows the results in a dialog box.
Open textwindow	Opens a text window to display the results.
Pass to operating system	Passes the result to the operating system (e.g. result of a `getfile` command)
Execute as Subquery	Opens a copy of the current session window and executes a DQL query.
Do nothing	No action is taken.

  (The option *Copy to Clipboard* is no longer supported. It was replaced by the variable %Clipboard% that allows more flexibility)

- Category: Functions can be grouped by assigning a category. A category name represents a submenu of the context menu, where all functions of this category are listed.

- Allow multiselect — Multiple objects can be selected and processed at the same time.
- Ask user before execution — The execution must be confirmed first.
- Active — Inactive functions are not visible in the context menu.
- Ignore errors — If an error occurs the execution is continued with the next object. Error messages will be displayed in the function result message after processing all objects.
- Create Debug Script: — This option is for function development only. Instead of executing the function, the resulting API script is opened in the API Script View, if the function was executed from a DQL Result List.

  Otherwise it will be opened in an extra text window and can be copied into any API Script view for execution.

  The generated debug script is full runnable in dqMan.
- DQL Result List — Function is available in DQL Result List
- Repository Navigator — Function is available in Repository Navigator
- Object Navigator — Function is available in Object Navigator

On the second page (not available for option “Execute as Subquery”) global variables can be defined. When the function is called, these variables are either filled by manual input of the user or queries from the repository using API commands. The values of the global variables are valid for all objects processed in one function call.

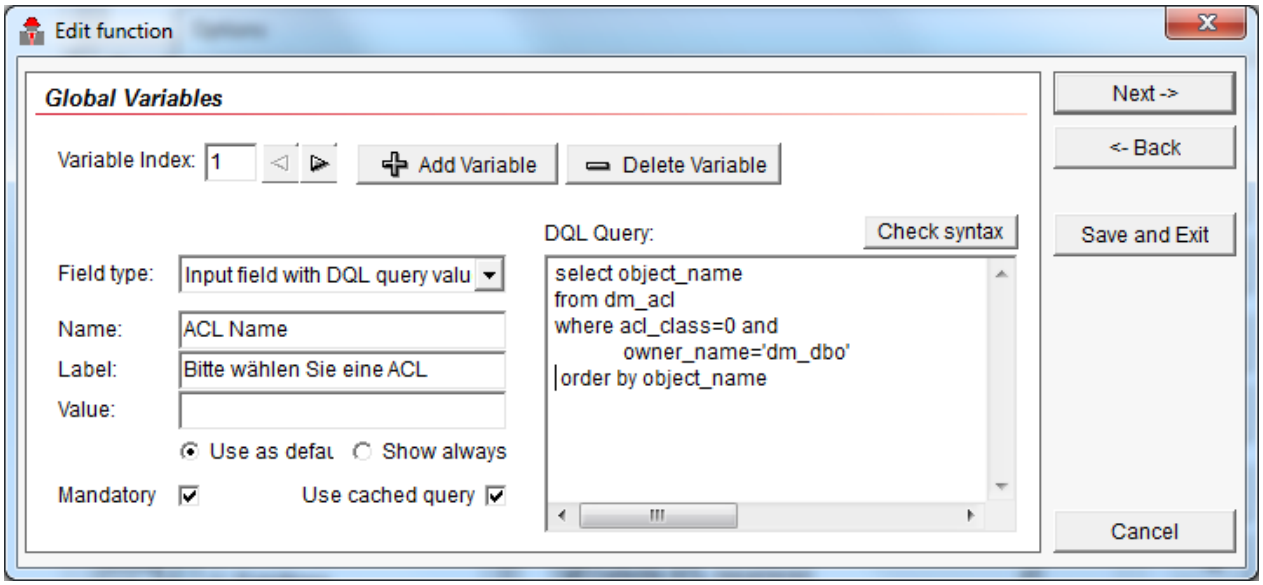


Figure 40 - User Function Assistant (2)

In the variables also column values of the DQL result list can be referenced (e.g. {r_object_id}). In this case the function can be executed for single objects only. The option “Allow multiselect” is deactivated automatically.

The DQL query:

```
Select full_format from dmr_content where any parent_id =
'{r_object_id}’ order by 1
```

results in a pick list that contains all available formats of an object. The column `r_object_id` must be contained in the DQL result list and only one object must be selected, otherwise this function is disabled.

The syntax of a DQL query can be checked by clicking on the button *Check syntax*, if a valid session is available in any session window.

Field type:

Input field:	Field where a user can/must input a value.
Input field with fixed valuelist:	An input field with fixed value assistance that must be entered into the control *Valuelist Data*. If items in the value list contain the characters || the part prior to || will be displayed in the dropdown control, the part after || will be returned as value. (e.g. Label||Value).
Input field with DQL query list:	An input field with value assistance that is filled with the results of a DQL query. The DQL query must be entered in the control *DQL Query*. If the select list of the DQL query contains more than one attribute, the value of the first column will be added to the value list, the value of the second column will be returned as variable value. The query `Select object_name, r_object_id from dm_document` fills the list with the `object_name` attribute of all dm_document objects. Selecting a name from the list returns the `r_object_id`.
Date picker:	Displays a date picker control.
File open dialog:	An input field with a browse button. A list of allowed file types can be entered in the control *File Formats*.
Directory selector:	Allows selection of a directory from the local file system.
Cabinet/Folder selector:	Allows selection of a cabinet or folder from the currently connected repository.
API Command:	A single line API command, like `id,c,dm_acl`. This command can reference result list columns by column names in curved clips, e.g. {r_object_id}. It can also reference global variables with a lower index than itself. (See function Change ACL for more information)

Name:	Name of the variable. Using this name the variable can be referenced in the function API sequence.
Label:	Caption displayed in the input dialog. If no label is available, the name will be displayed.
Default Value:	The default value shown in the input control. If the option ”Show always” is selected, this value will always appear, otherwise it is only valid until another value is entered or selected. This new value will then be displayed the next time the function is executed. It is allowed to reference columns of the DQL result grid by using standard syntax like {object_name}.
Mandatory:	Check this box, if it is a mandatory field.
Use cached query:	If checked, the result of the DQL query used to fill the value assistance will be cached to improve performance, but changes to the underlying values will be ignored. Select *Refresh Valuelists* from the *Session* menu to clear cache.

To add a new variable, click on the *Add variable* button. *Delete Variable* destroys the current variable.

The third page of the dialog either displays a textfield for the API sequence or for a DQL query (option" Execute as Subquery").

The Basic API Sequence Syntax

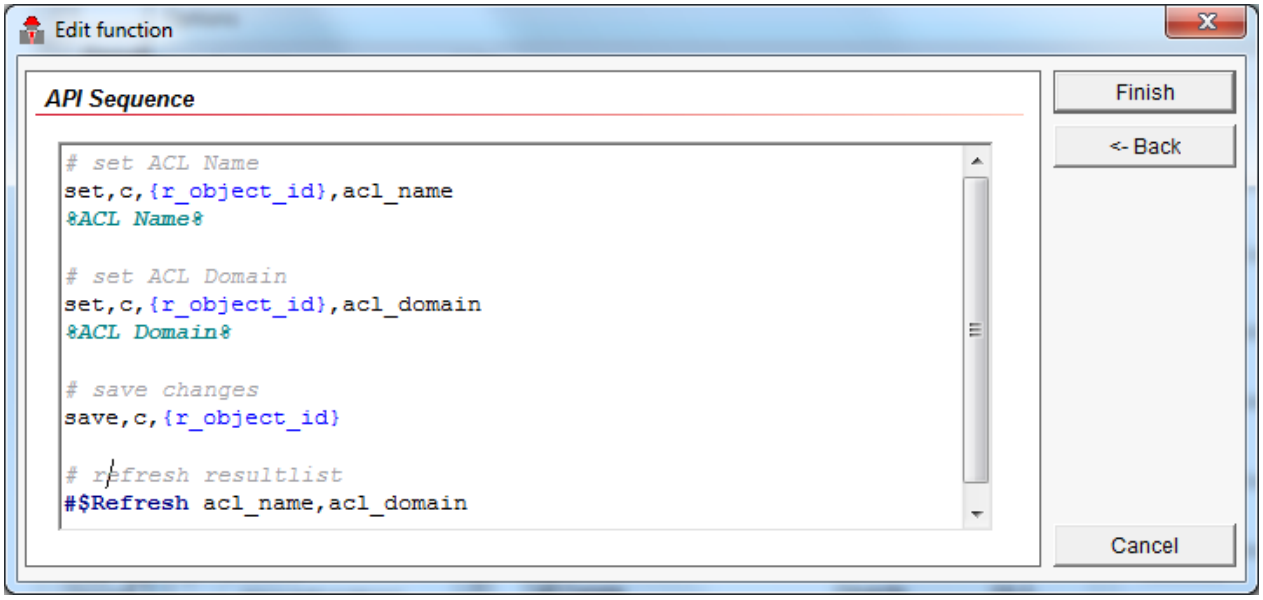


Figure 41 - User Function Assistant (3)

The API sequence consists of a batch of API commands, similar to an API script. References to values of the DQL result list are enclosed in curved clips. During execution, those references are replaced with values from the result list.

```
fetch,c,{r_object_id}
destroy,c,{r_object_id}
```

There are different placeholders to access the data of the result list

{*}	This is a reference to the currently selected column of the result list. If more than one column is selected, the function is disabled.
{* ID}	The selected column must contain a valid object id, otherwise the function is disabled.
{* INT}	The selected column must contain an integer value.
{* TIME}	The selected column must contain a valid time value.
{* CHAR}	The selected column must contain a string value.

Comments

Each line beginning with the # character or with // is regarded as comment line.

```
#fetch current object
fetch,c,{r_object_id}

# destroy the current object
destroy,c,{r_object_id}
```

Indenting

For functions, indenting is allowed for all kinds of instructions. All leading spaces and tabs will be removed prior to executing the command. For multiline commands the same amount of leading characters will be removed from all lines as was done for the first line, except any valid char is found.

```
#Set ACL Domain
  Set,c,{r_object_id},acl_domain
  %ACL Domain%
```

If debug scripts are created, all indents will be removed.

Variables

There are three different kinds of variables: global variables, local variables and predefined variables.

Global variables are defined in the object function assistant and filled once at function initialization. Their values are valid for all processed objects.

Local variables are used within the API sequence and are valid only for the processing of the current object.

Naming conventions:

The names of variables can be chosen freely, but it has to be enclosed with percentage signs. They are not case sensitive. Examples:

```
%1%
%File-Contents%
%this is a variable%
```

Assignments:

Constants or results of API commands can be assigned to a variable.

Examples:

```
%Repository_Owner%=dmadmin
```

initializes the variable `%Repository _Owner%` with the value " dmadmin".

(The `Define` command from earlier dqMan versions can still be used, but is marked deprecated)

```
%ID% = id,c,dm_document where object_name like 'a%'
```

assigns the result of the API command to the variable %ID%

Concatenating values:

Values can be concatenated in two ways:

```
%ID% =+ Add this text
%ID% = %ID%  Add this text
```

Usage:

To use a variable, insert it at the appropriate position in the API command:

```
get,c,%ID%,object_name
```

Predefined variables are:

%result%	This variable saves the result to be returned, e.g. `%result%=dump,c,{r_object_id}.` If %result% does not exist within the API sequence, the result of the last executed API command will be returned.
%CrLf%	This is a constant containing a carriage return and a line feed. It can be used for result formatting.
%Clipboard%	This variable allows to exchange text data with the clipboard. If this variable is used, the current content of the clipboard is pasted. If a value is assigned, it will be stored in the clipboard. Also concatenation is possible: `%Clipboard% = Any text %Clipboard% and more`

Example:

```
# set ACL name
set,c,{r_object_id},acl_name
%ACL Name%

# Set ACL Domain
set,c,{r_object_id},acl_domain
%ACL Domain%

# Save
save,c,{r_object_id}
```

Script Commands

Script commands extend the functionality of an object function. Placeholders or variables can be used as parameters. They will be replaced by their values before executing the script command.

Available script commands are:

#$EditFile <Filename>

Opens the currently processed object for editing and waits until the editor is closed.

#$Refresh <Column Name> {Column Name}

Refreshes the DQL result list by fetching the values of all attributes in the column list from the repository. If the char * was specified as column list, all currently selected columns are refreshed.

#$RemoveLine

Removes the currently processed line from the DQL result grid without destroying the Documentum object.

#$ShellExecute <Program> {Parameter}

Starts the <Program> with the supplied <Parameters> and returns control to the dqMan without waiting for its termination.

The first parameter is the executable program. The number of parameters is free. They must be separated by spaces.

#$ShellExecuteAndWait <Program> {Parameter}

Like #$ShellExecute, but dqMan waits for the program to finish its execution.

#$Sleep <time>

Pauses the function execution for <time> milliseconds.

#$Loop <collection identifier>

Start of a loop through a collection

#$Loop <attribute_name{(object_id)}>

Start of a loop through the values of the repeating attribute from the object identified by the ID object_id. If the object_id is not supplied the ID of the currently processed object from the result list is used.

#$EndLoop

Marks the end of the loop.

#$FetchObject <r_object_id>

Opens the Object Navigator and loads the object identified by r_object_id.

Example for a collection loop:

```
%Collection%=readquery,c,select group_name from dm_group where any
i_all_users_names = 'dmadmin'


#$Loop %Collection%
  %group_name% = get,c,%Collection%,group_name
#$EndLoop
```

The handling of the collection is done by the `#$EndLoop` command. It is not necessary to add next or close commands.

The variable `%LoopIndex<i>%` does only exist within a loop and contains the current loop index, starting with 0. `%Loopindex1%` is the index of the first loop. `%Loopindex2%` would be the index of the second loop, if it is nested within the first loop.

Example for an attribute loop

```
%id%= id,c,dm_document
#$Loop i_folder_id(%id%)
  %Folder_ID% =get,c,%id%,i_folder_id[%LoopIndex1%]
  #$Loop r_folder_path(%Folder_ID%)
   %FolderPath%=get,c,%Folder_ID%,r_folder_path[%LoopIndex2%]
  #$EndLoop
#$EndLoop
```

The variable `%LoopIndex<i>%` is incremented each time the `#$EndLoop` command is executed. It can be used as index for the repeating attribute values.

The DQL Query Syntax

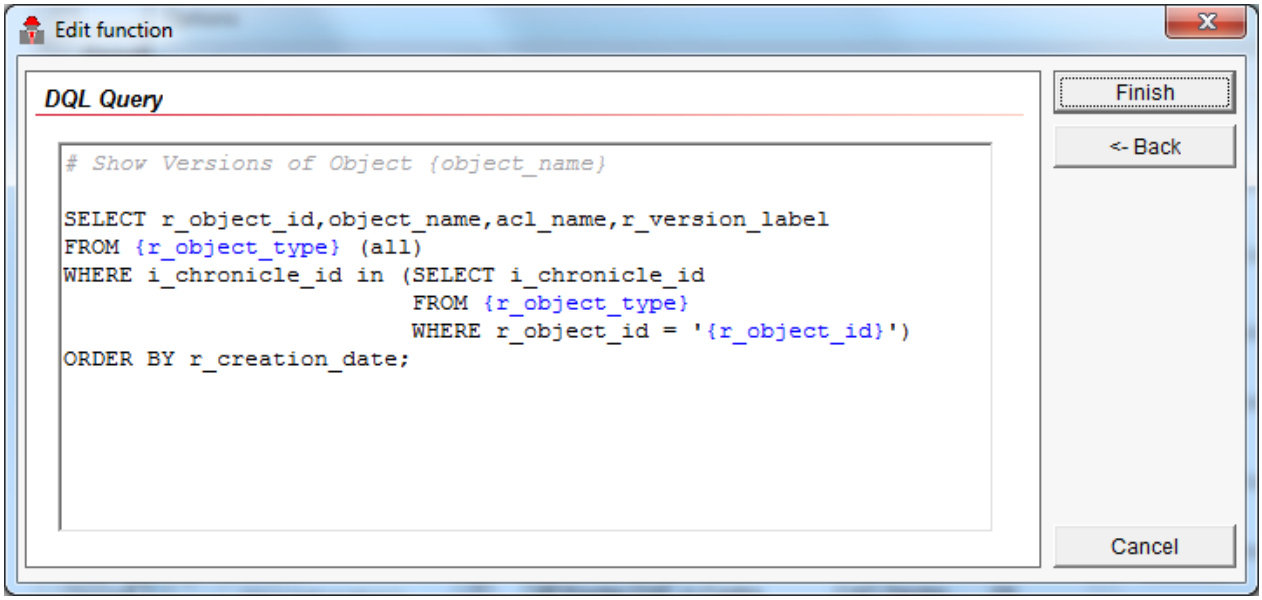


Figure 42 - User Function Assistant (4)

In the DQL query text area any DQL query can be inserted that is executable by dqMan. This includes comments, template fields etc.

To reference any values of the underlying DQL result list, insert the attribute names in curved brackets. If any referenced attributes are not available in the DQL result list, the function is disabled.

The multiselect option is not available for this type of function.

16.5.2. Session Functions

Session functions nearly offer the same elements as object functions.

One difference is that references to result list values, like {r_object_id}, are not allowed. The other is, that they are executed exactly once, independently from any selections made anywhere in the session window.

Installed session functions can be found in the Functions menu from the main menu. They are only available if the active window is a session window with a valid session.

16.6. Miscellaneous

16.6.1. Docbroker Availability

At startup, dqMan can check whether a docbroker host is available. If the docbroker host is not available, an error window is displayed which shows all in the dmcl.ini file configured docbrokers and their status.

There are different test modes available:

No check:	No checking.
Primary docbroker only:	Only the primary docbroker is checked and must be available. Backup docbroker will not be checked.
Any docbroker:	All available docbrokers (primary and backup) will be checked. At least one must be available.
Check all docbrokers:	All available docbrokers (primary and backup) will be checked. All must be available.

The button *Edit History* opens an editor to manage saved docbroker information.

16.6.2. Docbasic Interpreter

A valid Docbasic interpreter must be supplied to use the type *Docbasic Script* in the *Script Generator*. Default is DBEditor.exe that is installed with Documentum Developer Studio.

The generated script is saved as file and can be passed on to the Docbasic interpreter as parameter.

There are some predefined placeholders for parameters available:

{filename}: Filename of the script, generated by dqMan.
{user}: Username of the selected session window.
{password}: Password of the selected session window.
{docbase}: Repository of the selected session window.

The curved clips are a necessary part of the placeholders.

Example:

The Parameters:
{filename} -u{user} - p{password} -d{docbase}

will be transformed to:
DBEditor.exe c:\temp\script.ebs -udmadmin -padminpwd -dtestdocbase

16.6.3. Tracing

- Default trace level: Trace level that is used starting a trace.
- Always open tail viewer: Always opens a tail view, if a trace is started either by clicking on the *Trace* button of the toolbar or using the API command `trace,c,…`

17. Appendix A – Unsupported Commands and dqMan Syntax Extensions

This section holds information about undocumented DQL commands as well as dqMan syntax extensions to API and DQL that will not work in other environments.

17.1. The API Command: filename

This command can be used anywhere within dqMan (API command, API scripts, functions etc.). The usage is the same as the API get command, but it returns a valid windows filename. This is very helpful for all types of file operations, like export.

Within the returned value all invalid characters are replaced by an underscore.

Command	`get,c,<r_object_id>,object_name`	`filename,c,<r_object_id>,object_name`
Return value	`Name:Is this a slash / ?`	`Name_Is this a slash _ _`

17.2. The DQL Administration Method: set_session_dd_locale

This method can be used to change the current session locale with a DQL command or a DQL script (within dqMan only).

Syntax:

```
EXECUTE set_session_dd_locale WITH locale='new_locale'
```

`New_locale` must be a valid locale in the repository where the command is executed in, e.g. `en`.

dqMan uses this command in the generated scripts of the functions Edit Attribute or Attribute Create Script.

17.3. The DQL Administration Method: count_sessions

This is a regular, but undocumented DQL Administrative Method to find the details of sessions apart of list_sessions and show_sessions.

Syntax:

Execute count_sessions

Above DQL returns the following values:

concurrent_sessions	
history_sessions	
history_timeout	
hot_list_size	active sessions
warm_list_size	timed out sessions
cold_list_size	free sessions

This method is also offered through the session function "*Server Session Statistics*", available in the *Functions* menu.

17.4. Script Commands and Variables

Chapter *16.5 Functions* introduces script commands and the use of variables in API scripts. These are also dqMan specific and will not work anywhere else.

18. Appendix B - Plugins

Plugins expand the functionality of dqMan. They consist of a DLL, stored in the subdirectory "\Plugins" of the dqMan the program directory. The following two plugins are part of the dqMan installation.

18.1. The DMCL.INI / dfc Properties Editor

The editor offers fast access to the configuration files of Documentum. It is available from the *Edit* menu and opens the first file found following the environment path. Also the reference file dmclfull.ini / dfcfull.properties will be opened, if it is found.

The opened file depends on the currently used interface. If dmdl40.dll is used, dmcl.ini will be loaded, if dmcl.dll with dfc is used then dfc.properties will be shown.

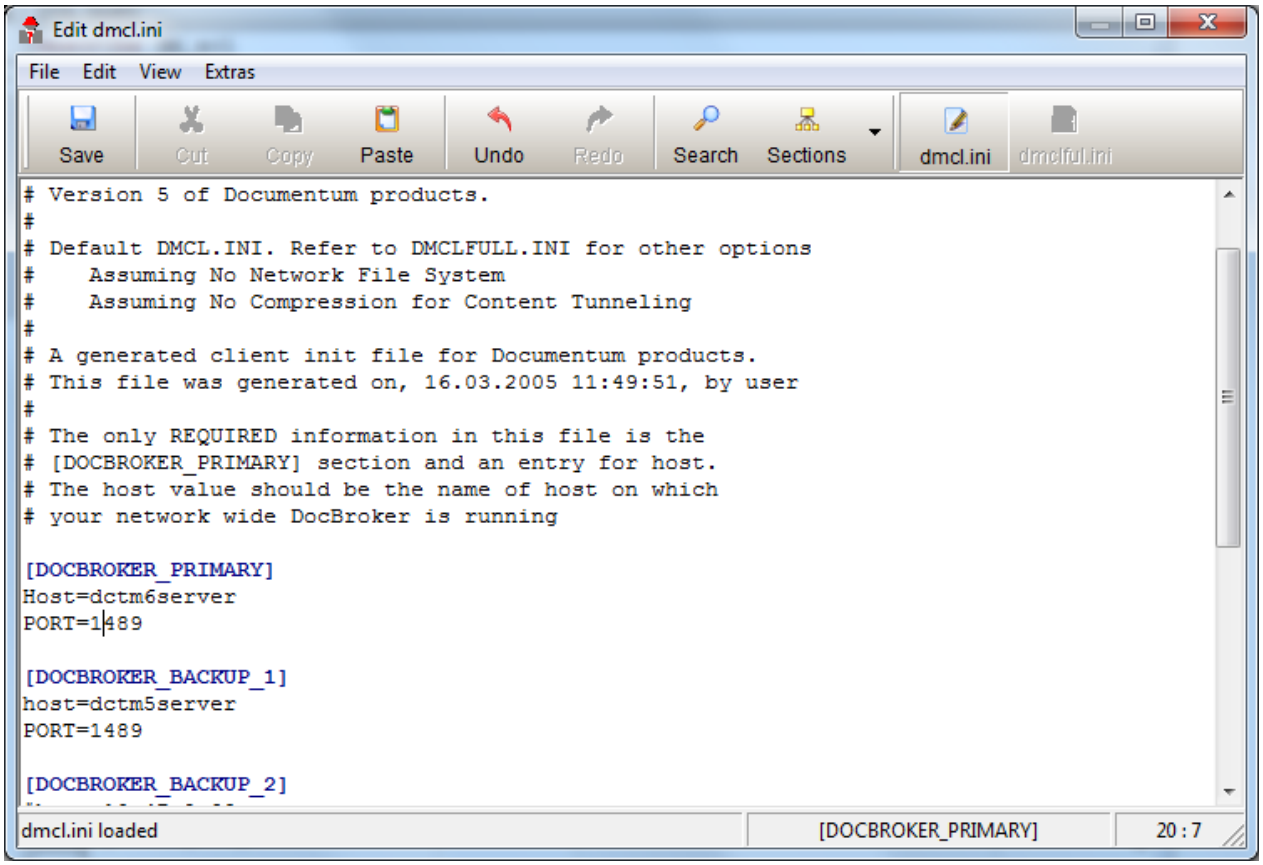

Edit dmcl.ini
File Edit View Extras
Save Cut Copy Paste Undo Redo Search Sections dmcl.ini dmclful.ini
Version 5 of Documentum products.
#
Default DMCL.INI. Refer to DMCLFULL.INI for other options
Assuming No Network File System
Assuming No Compression for Content Tunneling
#
A generated client init file for Documentum products.
This file was generated on, 16.03.2005 11:49:51, by user
#
The only REQUIRED information in this file is the
[DOCBROKER_PRIMARY] section and an entry for host.
The host value should be the name of host on which
your network wide DocBroker is running
[DOCBROKER_PRIMARY]
Host=dctm6server
PORT=1489
[DOCBROKER_BACKUP_1]
host=dctm5server
PORT=1489
[DOCBROKER_BACKUP_2]
dmcl.ini loaded
[DOCBROKER_PRIMARY]
20 : 7

Figure 43 - DMCL.INI Editor

For dmcl.ini, all entered docbroker hosts are stored in a host history. If at any time *host*= is entered or, while the cursor is positioned on an existing host name the keys <*Ctrl*> + <*Space*> are pressed, the history will be displayed for value assistance (dmcl.ini only).

The command *Clear host history* from the *Extras* menu can be used to clear the host history.

Changes to the dmcl.ini file will not take effect until at least all session windows of the dqMan are closed.
If the dfcfull.properties file is found, all available commands are loaded from this file and will be available as value assistance that will appear either if dfc. is entered or by pressing <*Ctrl*> + <*Space*>.

How and when changes to the dfc.properties file will take effect depends on the dfc behaviour. It may vary between different dfc versions.

18.2. The ACL Viewer

The ACL viewer displays ACLs for viewing and/or editing and is called either from the *View (→ View ACL)* menu or from the context menu of the DQL result list.

It is activated only if an active session is available and either the selected objects in the DQL result list are ACLs or an ACL is assigned to them (like dm_document or dm_folder) and the attribute `r_object_id` is available in the list.

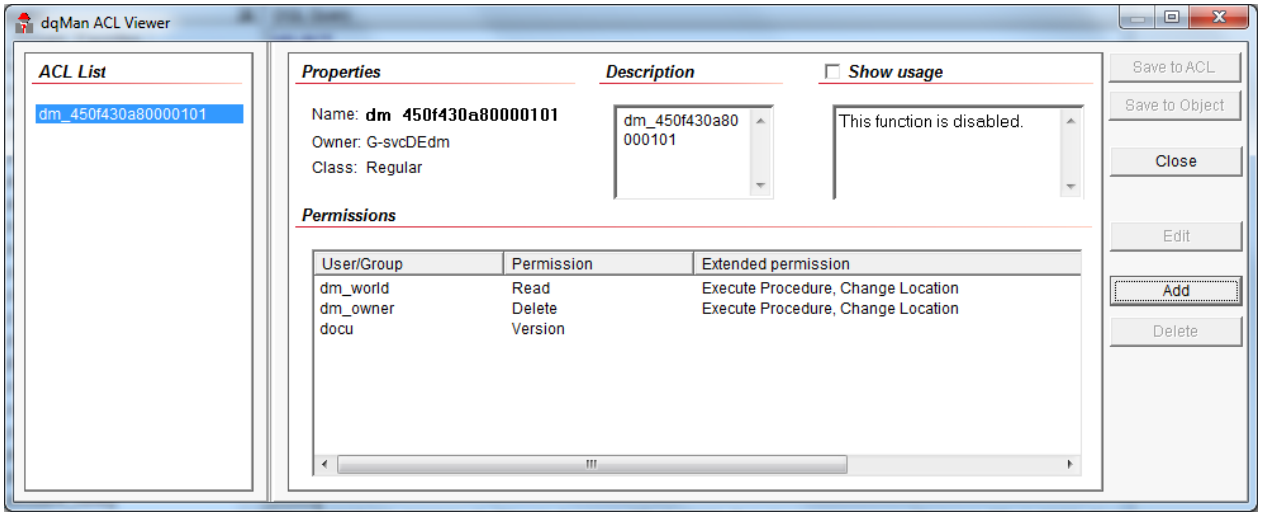


Figure 44 - ACL Viewer

The ACL list contains all ACLs from the selected objects of the DQL result list. To view or edit properties, an ACL can be selected from the list.

ACLs will only be added to the list once. Therefore, the number of selected objects can differ from the of the number of the ACLs in the list

The *Usage* area informs about the usage of the ACL in the current repository.

Change permissions

One or more selected permissions can be changed by clicking on the *Edit* button. If more than one permission was selected, only the changes will be applied, unchanged fields will be ignored. E.g. all groups can be assigned an extended permission without changing the basic permissions or the values of the other extended permissions.

Add permissions

The *Add* button opens a dialog to add one or more new permissions. If an existing permission was selected before, the settings of this permission will be copied to the dialog as default.

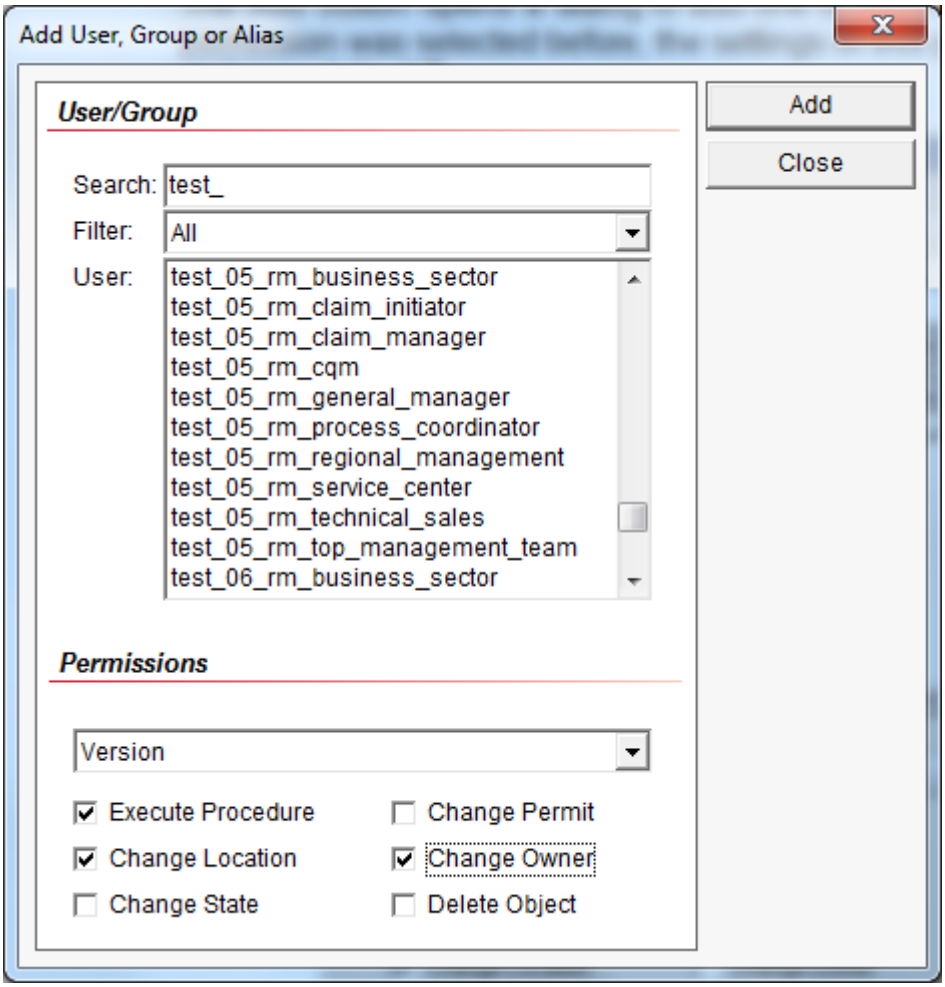


Figure 45 – Adding Permissions

To search for a group or user, an expression (begins with) can be entered into the *search* field. The *Filter* limits the displayed users and/or groups:

All:	All users and groups
User:	Users only
Group:	Groups only

Additionally available for Template ACLs:

Users and groups:	Users and groups
Qualified aliases:	Qualified aliases only
Unqualified aliases:	Unqualified aliases only

Multiple members can be selected from the list and will be added at once with identical permissions.

This procedure can be repeated until either the list of the users is empty or the dialog is closed.

Delete permissions

Selected permissions can be deleted clicking on the *Delete* button.

Save changes

Changes can be saved in two different ways:

Save to ACL: Saves the changes to the ACL object. All objects that have been assigned this ACL are affected.

Save to Object: Saves the changes to the object only, i.e. a new ACL will be created and assigned to the objects selected in the DQL result list. This function is only available if the object from the DQL result list is not an ACL itself. If more than one object with the same ACL is selected in the DQL Result List, a new ACL will be created for each of them.

View group members

If permissions are assigned to group names, the contained users and their resulting permissions can be displayed with the function *Show users* from the context menu. More than one group can be selected.

In that list all users are displayed, together with their group names, the highest resulting basic permission as well as the aggregated extended permissions.

Copy permissions

Selected permissions can be copied between different ACLs using the functions Copy, Cut and Paste from the context menu. The behaviour is the same as using the common clipboard functions.

If a permission is inserted into an ACL which grants permissions to already existing users or groups, several options are available:

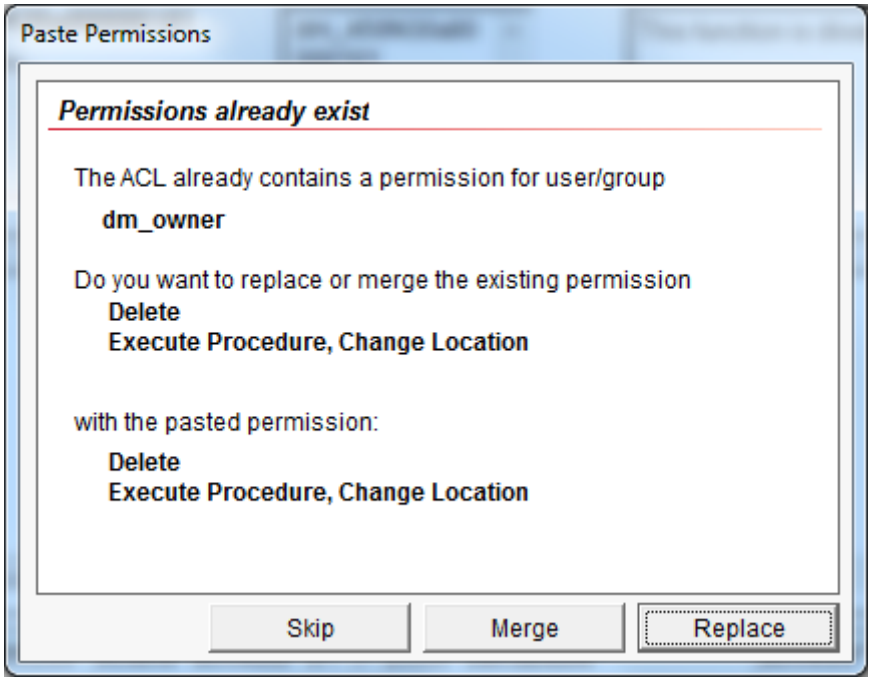


Figure 46 – Copy Permissions

Replace: The old permission is deleted and replaced by the new one.

Merge: Permissions are merged, meaning the higher basic permission is assigned and the extended permissions are aggregated.

Skip: This permission will not be pasted.

18.3. dqMan Analyzer

dqMan Analyzer is intended to provide a simple and fast way to generate regular analysis of repository data. The plugin itself is used to execute a batch of predefined queries. The results are then stored to an Excel template which does the analysis.

An analysis consists of two files, an XML where the queries are described as well as an Excel template. It is very useful to set the read-only flag for the Excel template to prevent overwriting it by accident.

The files must be stored in the /Analyzer sub directory of the dqMan home directory (c:\program files\dqMan). All XMLs will be read on start-up and all available analysis reports will be placed in an additional main menu called *Analyzer*.

There are two preinstalled examples available, a simple (UserStatistics) and a complex one (StorageStatistics).

The XML has the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data>
  <analyzer>
    <name>User Statistics</name>
    <label>User Statistics</label>
    <template>UserStatistics.xls</template>
    <repositories>Comma separated repository list</repositories>
    <query>
      <worksheet col="3" row="3" >Table of Contents</worksheet>
      <api>get,c,docbaseconfig,object_name</api>
    </query>
    <query>
      <worksheet col="3" row="5" >Calculations</worksheet>
      <dql>select count(client_capability) from dm_user
           group by client_capability order by
           client_capability</dql>
    </query>
  </analyzer>
</data>
```

The tags are:

<name>: Internal name.

<label>: The label used for the Analyzer menu.

<template>: Name of the Excel template without path, must be stored in the /Analyzer subdirectory.

<repositories>: Comma separated repository restriction list. If empty, this anaylsis is available in all repositories.

<query>: Element of the query list.

<worksheet>: Name of the worksheet where the data is pasted, col and row define the cell where the data resides. If a list of values is queried, col and row indicates the top left cell of the list.

<dql>: A DQL query to execute. DQL queries can contain template fields to provide more flexibility.

<api>: An API command to execute.

18.4. Content Restore

The Content Restore template allows the restoring of deleted non content-less objects (usually dm_document and its descendants).

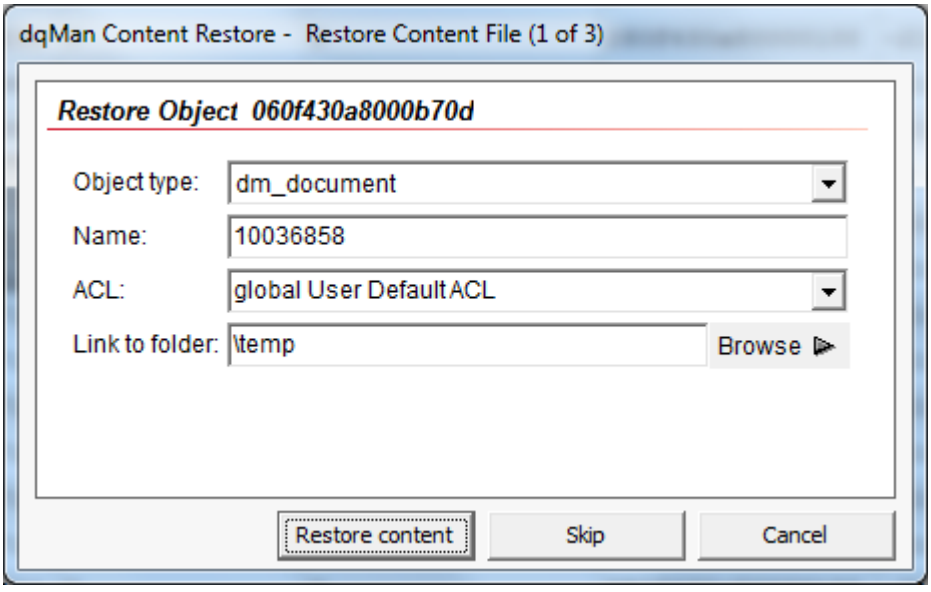
The plugin adds two functions to the context menu of the DQL result list. They are available, if the list contains dmr_content objects.

Example query:

```
Select * from dmr_content where parent_count=0;
```

Content File: View
This function opens the content of the dmr_content object for viewing. It is allowed for all objects in the list, regardless if they are deleted (parent_count=0) or are still linked to a dm_sysobject.

Content File: Restore
A dialog is opened to allow some settings for the restore. The name is prefilled with the original filename taken from the attribute setfile. ACL and folder link is not mandatory.



A click on *Restore content* will create a new object of the selected type and place it in the selected folder. Attributes cannot be restored for technical reasons.

If more than one object is selected to be restored, the dialog is loaded with the data for the next object.